Computer Vision

Local Features

Luis F. Teixeira & Jorge Silva | M.EIC | FEUP

Previous topics

- Edge detection
  - Sobel (OpenCV -> Sobel() and Scharr())
  - Canny (OpenCV -> Canny())
  - Laplacian (OpenCV -> Laplacian())
- Hough transform
  - lines (OpenCV -> HoughLines() and HoughLinesP())
  - circles (OpenCV -> HoughCircles())

Next topics

- Local features
  - feature detection
    - corners: Harris corner detector
    - blobs: SIFT (Scale-Invariant Feature Transform)
    - ref. to other detectors
  - feature description
    - SIFT
    - ref. to other descriptors
  - feature matching
    - distance measures
    - removing outliers
    - brute-force & FLANN (Fast Library for Approximate Nearest Neighbors)

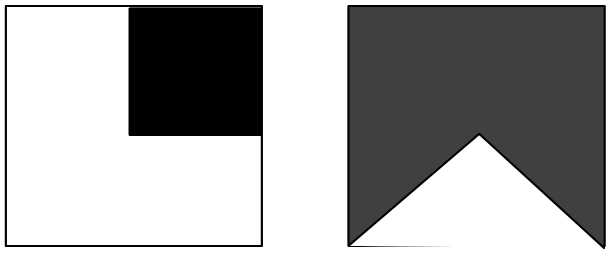
What are local features ?

- Local feature
  - an image pattern
    which differs from its immediate neighborhood
- Image properties commonly considered:
  - intensity, color, texture
- Local features can be:
  - points, corners, edgels, small image patches
- Typically, some measurements are taken from a region centered on a local feature and converted into descriptors

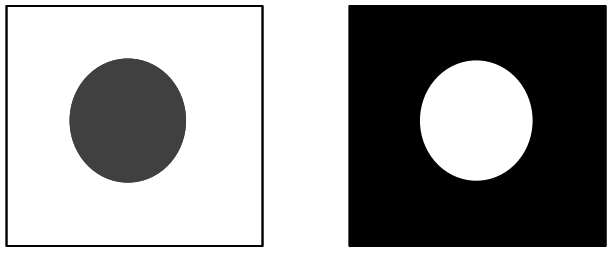
NOTE: in these slides "feature point" , "keypoint" and "interest point" are used interchangeably.

Feature points: corners *vs* blobs

- Feature point
  - is any point that can be located robustly,
    which includes corners but also includes such features as small marks
- Corners
  - technically, a corner is the intersection of two edges
- Blobs
  - regions in a digital image that differ in properties,
    such as brightness or color, compared to surrounding regions



Corners



Blobs

Local features - example

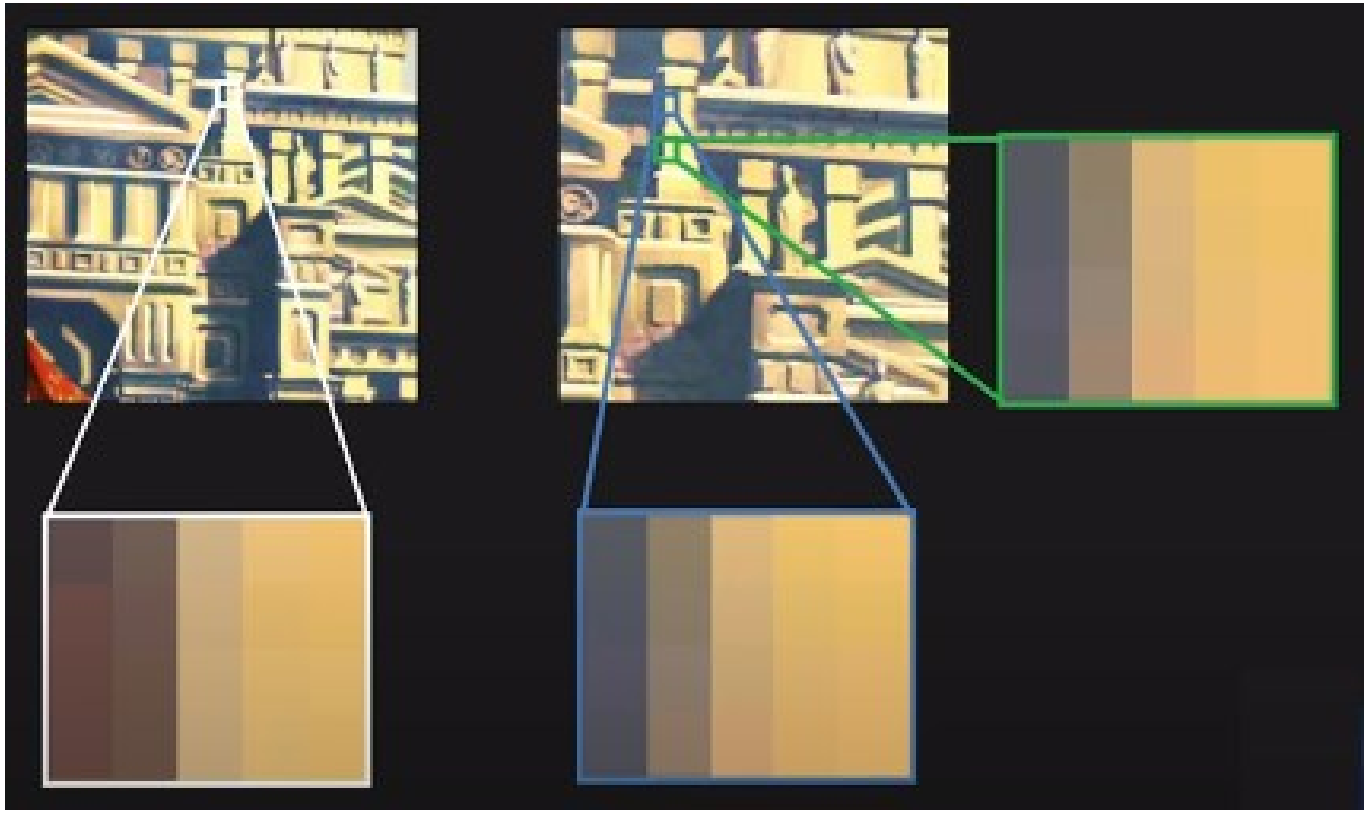
Corners (Harris corner detector)

Blobs (SIFT detector)

https://docs.opencv.org/3.3.0/da/df5/tutorial_py_sift_intro.html

- Feature points are signaled by circles
  - some feature points are not quite obvious … !!!
- Feature scale (*see next slides*) represented by the size of the circle

Are edges interesting?



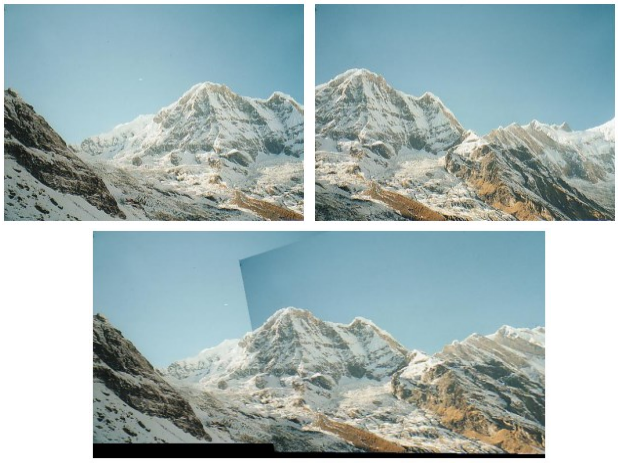
slide: adapted from Shree Nayar, Columbia University

– Edges may have a semantic interpretation
  - ex: edges in aerial images may correspond to roads

but they are not interesting points
  - they are not descriptive enough and unique enough (cannot localize and edge!)

Why use local features ?

- Local (invariant) features are a powerful tool
- Possible usage examples:
  - They provide a set of well localized and individually detectable anchor points (what the features represent is not actually relevant) that can be used for:
    - object recognition
    - indexing and database retrieval
    - image alignment / stitching
    - camera calibration and 3D reconstruction
    - tracking
    - robot navigation



  - They may have a semantic interpretation
    - ex: blobs in an tissue image may correspond to impurities
  - They can be used as robust image representation
    - **object or scene recognition without the need for segmentation**

Ideal local features

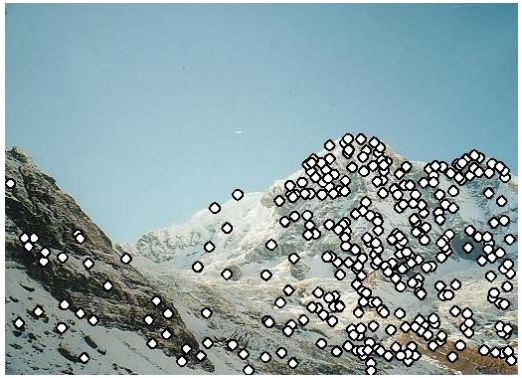
- Ideally,
  one would like such local features to correspond to
  semantically meaningful object parts
- In practice, however, this is unfeasible,
  as this would require
  high-level interpretation of the scene content,
  which is not available at this early stage
- Instead,
  detectors select local features directly
  based on the underlying intensity patterns

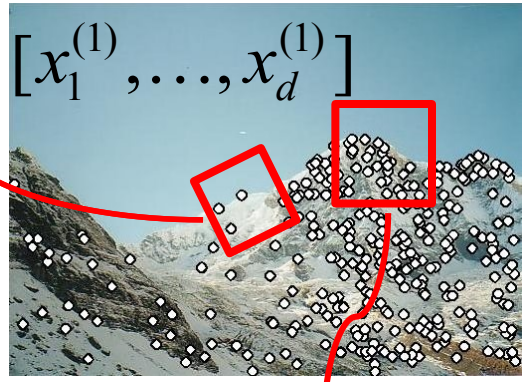
Local features: main processing steps

- **Detection**:
  Identify the interest points

- **Description**:
  Extract feature descriptor (vector) surrounding each interest point.

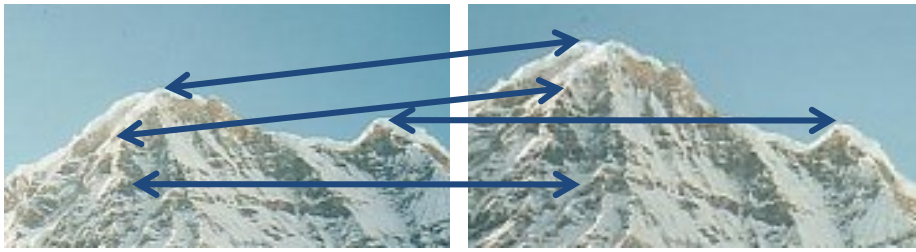
- **Matching**:
  Determine correspondence between descriptors in two views



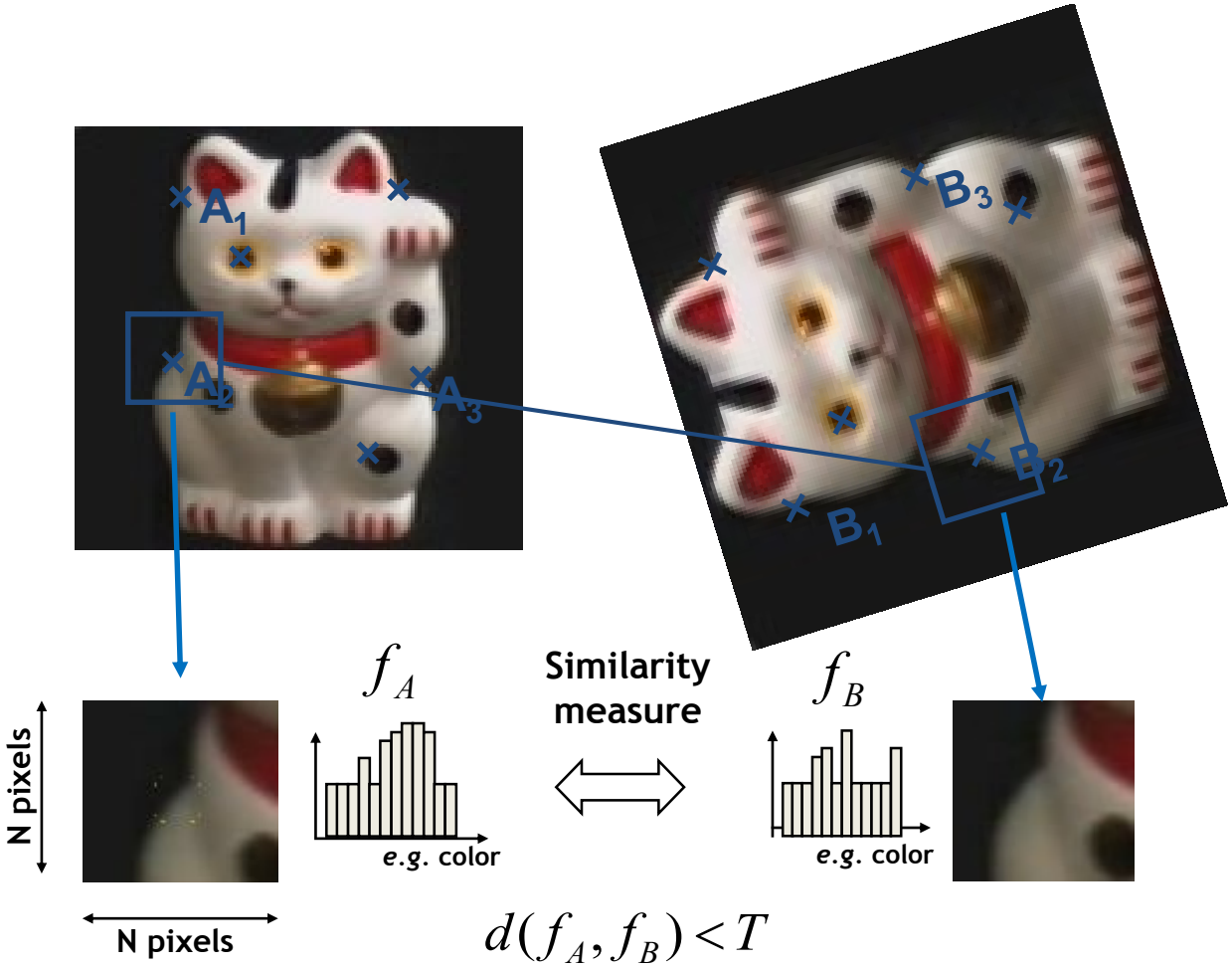


$$\mathbf{x}_1 = [x_1^{(1)}, \ldots, x_d^{(1)}]$$

$$\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$$



Local features: main processing steps



1. Detect a set of distinctive keypoints
2. Define a region around each keypoint
3. Extract and normalize the region content
4. Compute a local descriptor from the region
5. Match local descriptors

Local features: desired properties

- Locality
  - A feature occupies a relatively small area of the image
- Saliency / Distinctiveness
  - Each feature has a distinctive description
- Repeatability
  - The same feature can be found in several images despite geometric and photometric transformations
    - We want to detect (at least some of) the same points in the images.
- Quantity
  - We need a sufficient number of features to cover the image, but …
- Compactness and efficiency
  - Many fewer features than image pixels
  - Should allow for time-critical applications

Feature point detectors & descriptors

- Desired properties
  - Invariant to photometric variations
  - Invariant to geometric transformations (translation, rotation, scaling, … … Invariant to viewpoint)

Feature point detectors

- Corner detectors
  - "corner score" is derived from the 2nd-order moment image gradient matrix
    - Harris & Stephens (1988), (https://docs.opencv.org/4.x/dc/d0d/tutorial_py_features_harris.html)
      Shi-Tomasi detectors (1994) (https://docs.opencv.org/4.x/d4/d8c/tutorial_py_shi_tomasi.html)
  - Features from Accelerated Segment Test (FAST) corner detector
    (https://docs.opencv.org/4.x/df/d0c/tutorial_py_fast.html)
- Blob detectors
  - use local extrema of the responses of certain filters as interest points
    - Laplacian of Gaussian applied at multiple image scales (Lindberg, 1994)
    - Local extrema of an image filtered with differences of Gaussians (Lowe, 1999, 2004)
    - Fast Hessian detector (Bay et al., 2008)
- Affine-invariant detectors
  - detectors that are invariant to affine changes (different scaling in different directions)
  - provide higher repeatability for large affine distortions
    - Lowe (2004); Mikolajczyk and Schmid (2002); Matas et al. (2002)
  - note: not covered in this course but available in OpenCV library
    (https://docs.opencv.org/4.x/d3/d28/classcv_1_1MSER.html)

NOTE: usually non-maximum supression is applied to detector results

- this has the effect of supressing all interest points that are not local maxima

Harris corner detector

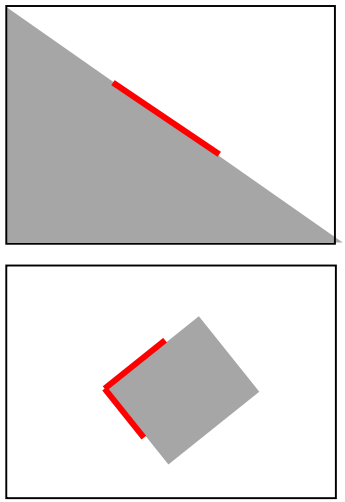
- **Basic idea:**
  - find points where two edges meet—
    i.e., high gradient in orthogonal directions
- **Harris corner detector**
  - Allows the calculation of the "strength of a corner",
    for each pixel, *p*, of an image window
  - The calculation is based on the gradient inside the window

  - **window gradient =** $[E_x\ E_y]^T$,
    where E_x E_y are the spatial derivatives
    - $E_x = \partial E / \partial x$; $E_y = \partial E / \partial y$
    - calculated using, for example, Sobel filters
  - *p* – image pixel
  - *Q* – neighborhood of (2*N*+1) x (2*N*+1) *pixels*, around *p*
  - *C* – the following matrix, calculated in the neighborhood *Q* **of** *p*

$$C = \begin{bmatrix} \sum E_x^2 & \sum E_x E_y \\ \sum E_x E_y & \sum E_y^2 \end{bmatrix}$$

Harris corner detector

- Matrix *C* caracterizes the graylevel "structure".
- Using the eigenvalues and eigenvectors of *C*
  it is possible to detect the existence of corners in the windows
  - let λ_1 and λ_2 be the eigenvalues of *C* (both positive)
  - if the intensity of the pixels inside the window is approximately constant, then
    - $\lambda_1 \approx \lambda_2 \approx 0$
  - if the window contains an ideal step edge
    (transition black↔white) we have:
    - $\lambda_1 > 0$ and $\lambda_2 = 0$
    - eigenvector associated with λ_1 is parallel to the gradient (perpendicular to the edge)
  - if the window contains a corner, then:
    - $\lambda_1 >= \lambda_2 > 0$
    - the greater the λ's, the greater the contrast
    - eigenvectors are parallel to the gradients
- In conclusion:
  - eigenvectors of *C* codify edge orientation
  - eigenvalues of *C* codify edge magnitude



Eigenvalues and eigenvectors

(

Eigenvalues
and
Eigenvectors

Eigenvalues and eigenvectors

- Consider that
  - A is a square matrix
  - v is a column vector
  - λ is a scalar
- **If** $A.v = \lambda v$
  - assuming a non-trivial solution ($v \neq 0$)
  - v is an *eigenvector* **of** A
  - λ is an *eigenvalue* **of** A

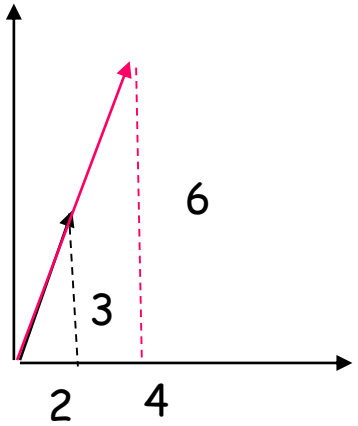
(http://mathworld.wolfram.com/Eigenvalue.html)

Eigenvalues and eigenvectors

- Example:

$$A = \begin{bmatrix} 5 & -2 \\ 6 & -2 \end{bmatrix}$$

$$v = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$



$$Av = \begin{bmatrix} 5 & -2 \\ 6 & -2 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix} = 2 \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

v is an eigenvector with eigenvalue 2

Eigenvalues and eigenvectors

$$Av = \lambda v$$

$$Av - \lambda v = 0$$

$$(A - \lambda I)v = 0$$

homogeneous system of linear equations

v is an eigenvector if it is not the trivial solution, *v*=0;

according to Cramer's rule,
a system of linear equations has non-trivial solutions
if and only if the determinant is zero

$$det(A - \lambda I) = 0$$

Eigenvalues and eigenvectors

- Example:

$$A = \begin{bmatrix} 5 & -2 \\ 6 & -2 \end{bmatrix}$$

$$\det(A - \lambda I) = \det \begin{vmatrix} 5-\lambda & -2 \\ 6 & -2-\lambda \end{vmatrix} = 0$$

$$(5-\lambda)(-2-\lambda) - (6)(-2) = 0$$

$$\lambda^2 - 3\lambda + 2 = 0 \;\Rightarrow \lambda = \frac{3 \pm \sqrt{9-8}}{2}$$

$$\lambda_1 = 1, \lambda_2 = 2$$

Eigenvalues and eigenvectors

- Eigenvector for λ=1

$$\begin{aligned} (5-1)v_{11} - 2v_{12} &= 0 \\ 6v_{11} + (-2-1)v_{12} &= 0 \end{aligned} \quad \Rightarrow \quad \begin{aligned} 4v_{11} - 2v_{12} &= 0 \\ 6v_{11} - 3v_{12} &= 0 \end{aligned}$$

$$\Rightarrow 2v_{11} = v_{12} \Rightarrow \quad v_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Eigenvalues and eigenvectors

- Eigenvector for λ=2

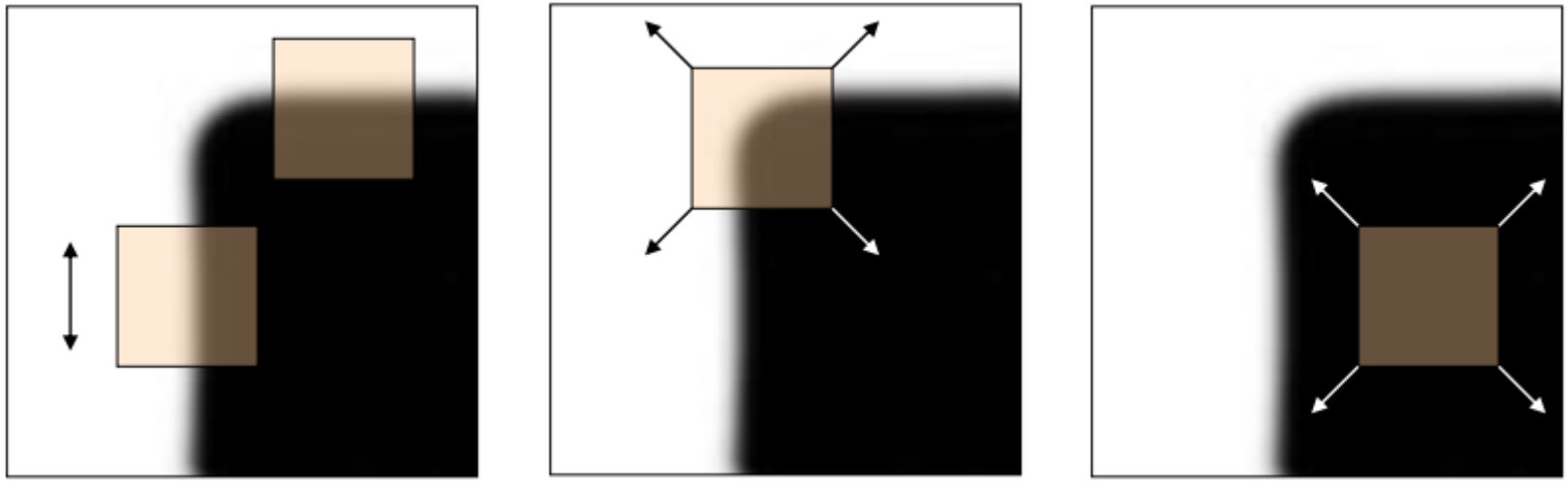
$$\begin{aligned} (5-2)v_{21} - 2v_{22} &= 0 \\ 6v_{21} + (-2-2)v_{22} &= 0 \end{aligned} \quad \Rightarrow \quad \begin{aligned} 3v_{21} - 2v_{22} &= 0 \\ 6v_{21} - 4v_{22} &= 0 \end{aligned}$$

$$\Rightarrow 3v_{21} = 2v_{22} \quad \Rightarrow \quad v_2 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

Eigenvalues and eigenvectors

)

Harris corner detector



$\lambda_1 >> \lambda_2$ or $\lambda_2 >> \lambda_1$ | λ_1 and λ_2 are large | $\lambda_1 \approx \lambda_2 \approx 0$

One way to score cornerness / corner strength):

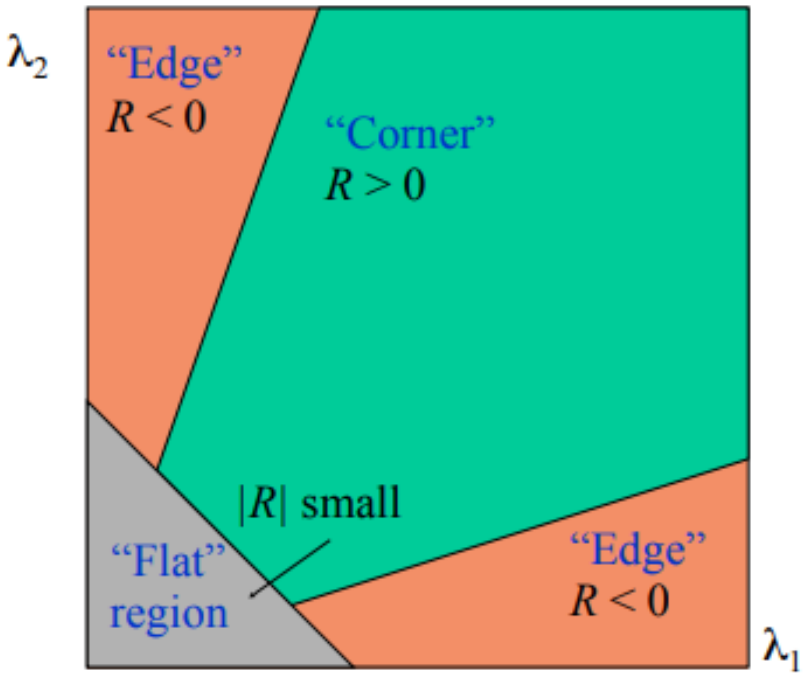
$$\mathbf{R = \lambda_1\lambda_2 - k(\lambda_1+\lambda_2)^2} \quad \mathbf{,\ k} = \text{constant}$$

Procedure:

1) Compute **C** matrix for window surrounding each pixel to get its cornerness score (R).
2) Find points with large score (R > threshold)
3) Take the points of local maxima (non-maxima suppression)

Harris corner detector

$$R = \lambda_1\lambda_2 - \alpha(\lambda_1+\lambda_2)^2$$

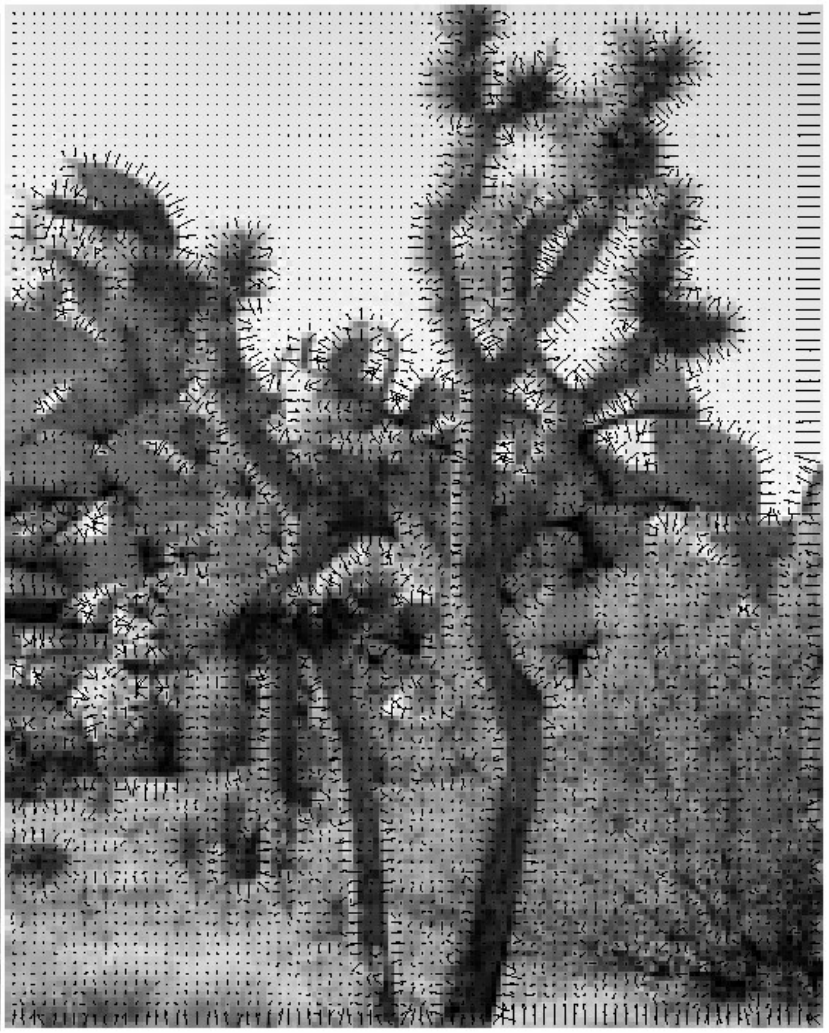


OpenCV: cv2.cornerHarris() and cv.cornerSubPix() - note: corners can be found with subpixel accuracy

https://docs.opencv.org/4.x/dc/d0d/tutorial_py_features_harris.html

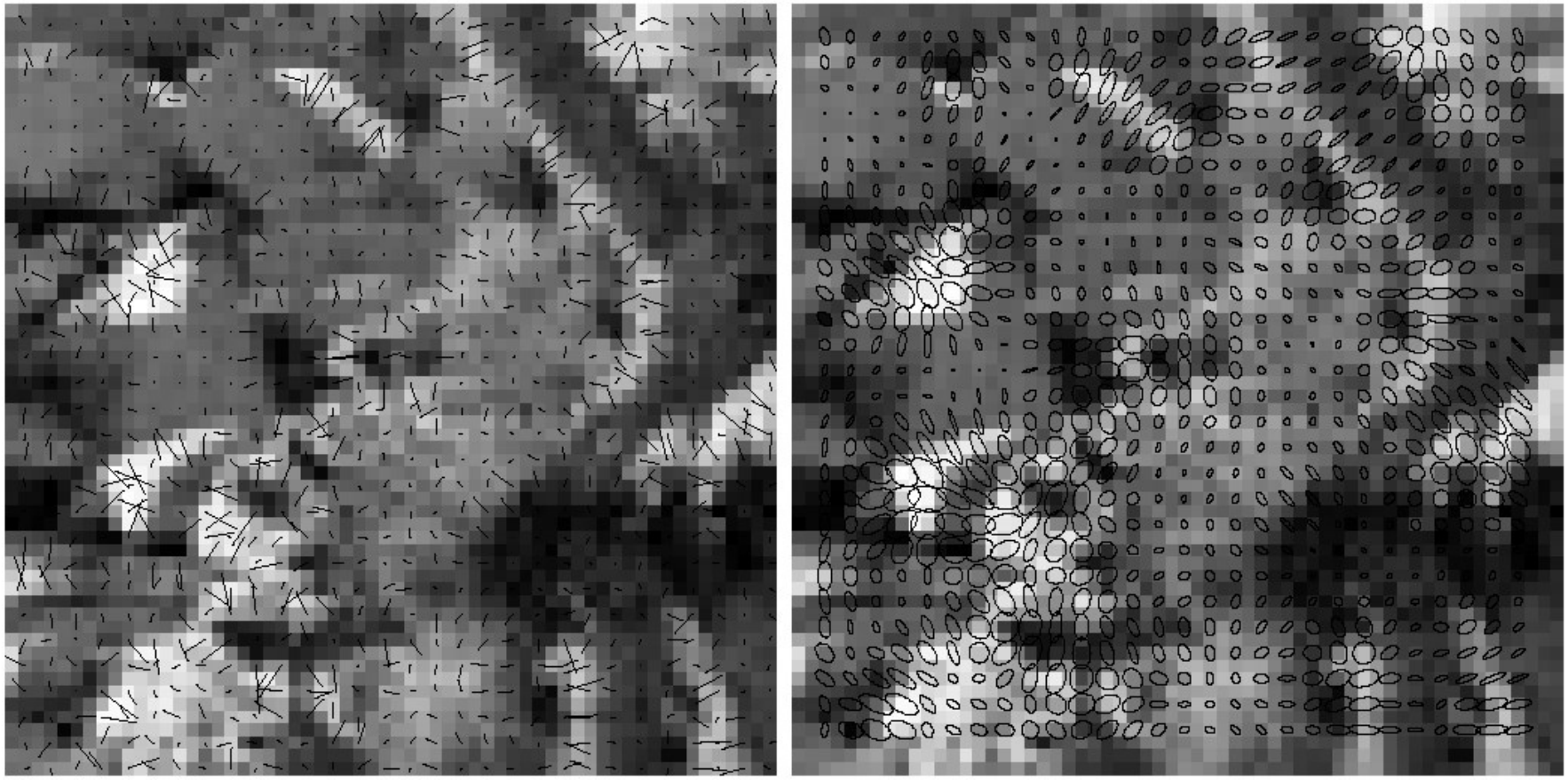
Harris corner detector

Original



Gradient

Harris corner detector

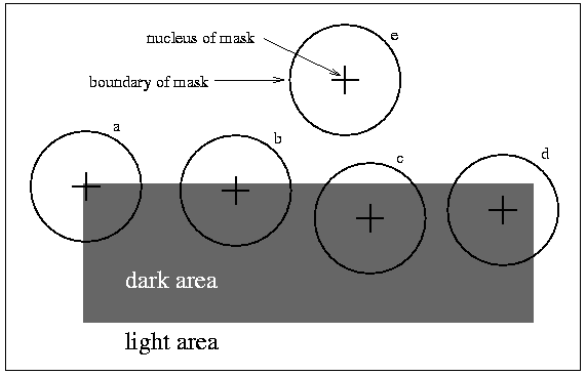


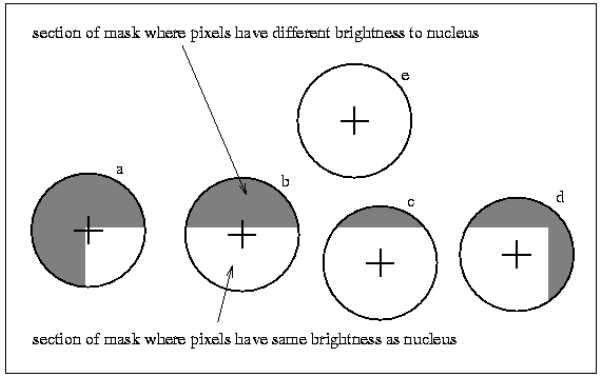
Zooming of a region of the previous image.
The ellipses indicate the eigenvalues and eigenvectors of the *C* matrices

Other corner detectors

- Other corner detectors (http://www.ee.surrey.ac.uk/Research/VSSP/demos/corners/survey.html)
  - Modern variants of the Harris corner detector
    (see Szeliski book)
  - Shi-Tomasi corner detector (OpenCV: cv.goodFeaturesToTrack())
    (see Szeliski book)
  - FAST corner detector (OpenCV: https://docs.opencv.org/4.x/df/d0c/tutorial_py_fast.html)
  - Moravec detector (one of the first corner detectors)
    (http://rfv.insa-lyon.fr/~jolion/PAPIERS/ptint/node2.html)
  - SUSAN (Smallest Univalue Segment Assimilating Nucleus) detector
    (http://www.fmrib.ox.ac.uk/~steve/susan/)

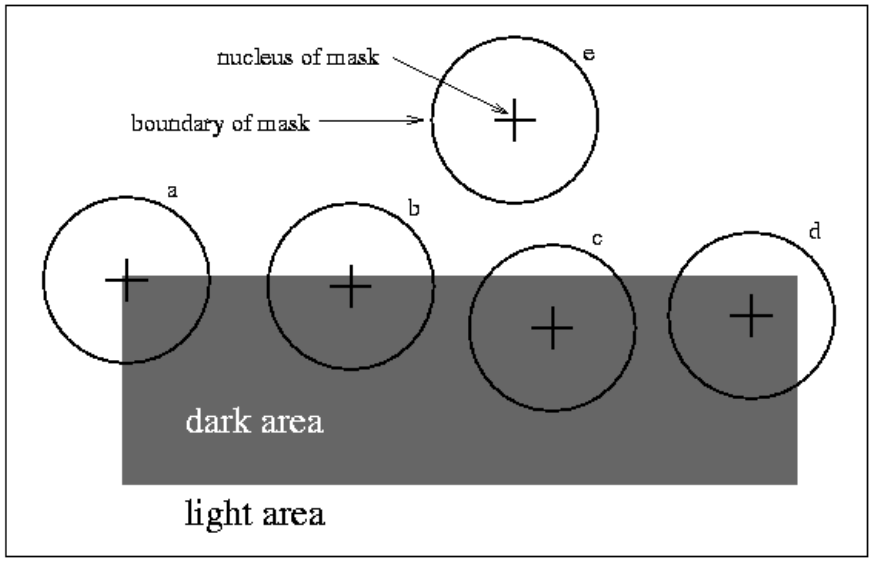
SUSAN
- circular masks
- USANs are the white parts similar to the nucleus (cross)
- **minimum values of the USANs occur at the corners**

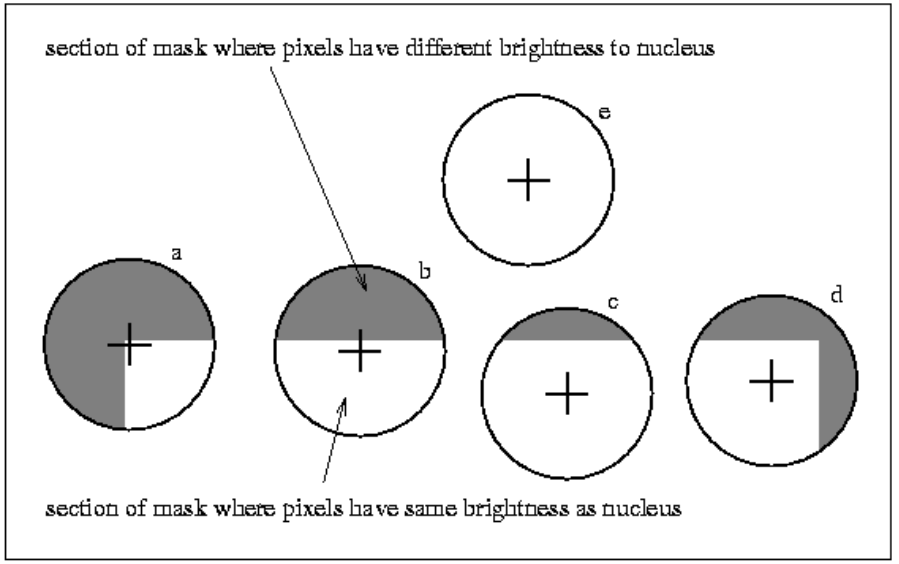




SUSAN corner detector

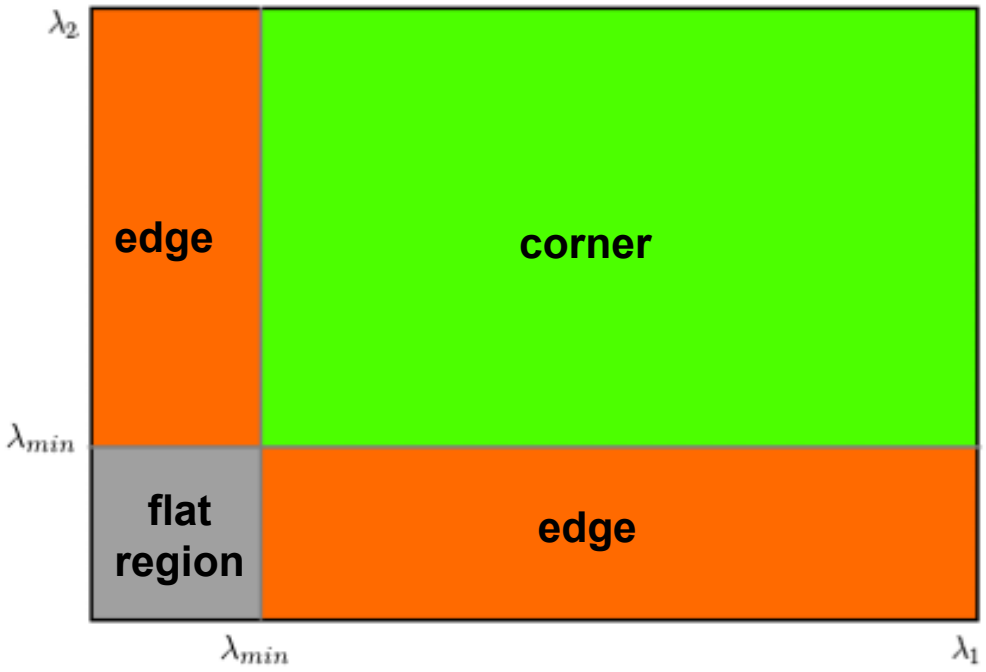
- SUSAN (Smallest Univalue Segment Assimilating Nucleus) detector (http://www.fmrib.ox.ac.uk/~steve/susan/)
- circular masks
- USANs are the white parts similar to the nucleus (cross)
- minimum values of the USANs occur at the corners





Shi-Tomasi corner detector

$$R = \min(\lambda_1, \lambda_2)$$

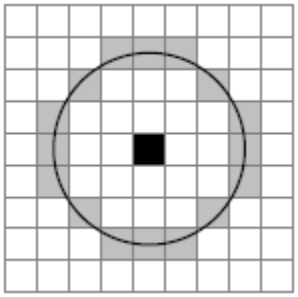


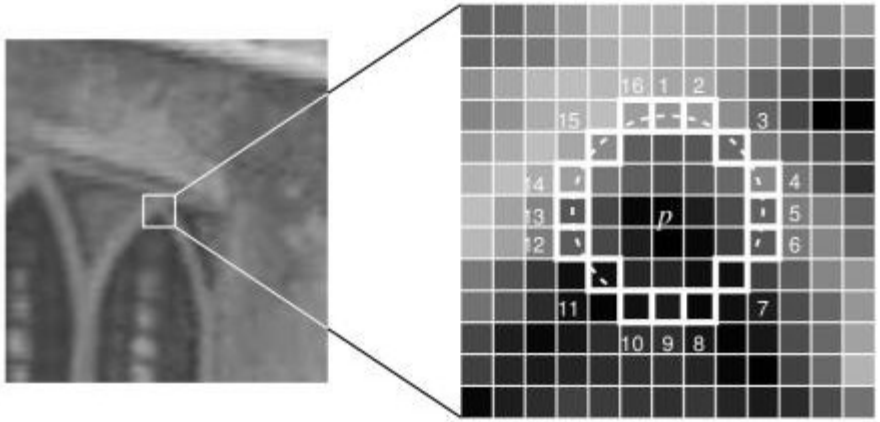
OpenCV: cv2.goodFeaturesToTrack().

https://docs.opencv.org/4.x/d4/d8c/tutorial_py_shi_tomasi.htmll

FAST corner detector

- Features from Accelerated Segment Test (FAST) - Rosten and Drummond (2005, 2006)
- A highspeed corner detector.
- Select a pixel **p** in the image which is to be identified as an interest point or not. Let its intensity be $\mathbf{I_p}$.
- Select appropriate threshold value **t**.
- Now the pixel **p** is a corner if there exists a set of **n** contiguous pixels in the **circle (of 16 pixels)** which are all brighter than $\mathbf{I_p + t}$, or all darker than $\mathbf{I_p - t}$. (Shown as white dash lines in the image below; **n** was chosen to be 12).
- A high-speed test was proposed to exclude a large number of non-corners. (*see* https://docs.opencv.org/4.x/df/d0c/tutorial_py_fast.html *for details*)

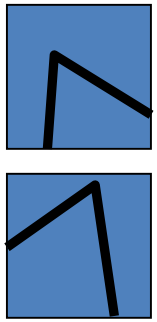




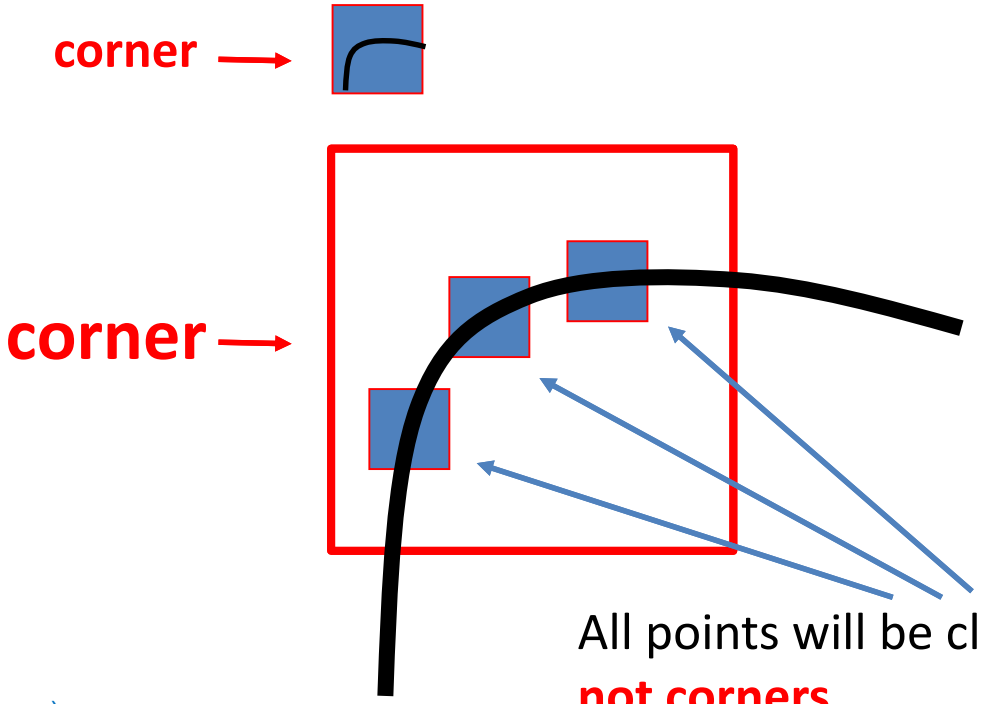
Corner detectors (variant to scale)

- Ex: original* Harris properties
  - Invariant to rotation

    i.e. eigenvalues remain the same



  - Not invariant to image scale



(*) Note:
scale-invariant versions of Harris corner detector have been proposed in the literature

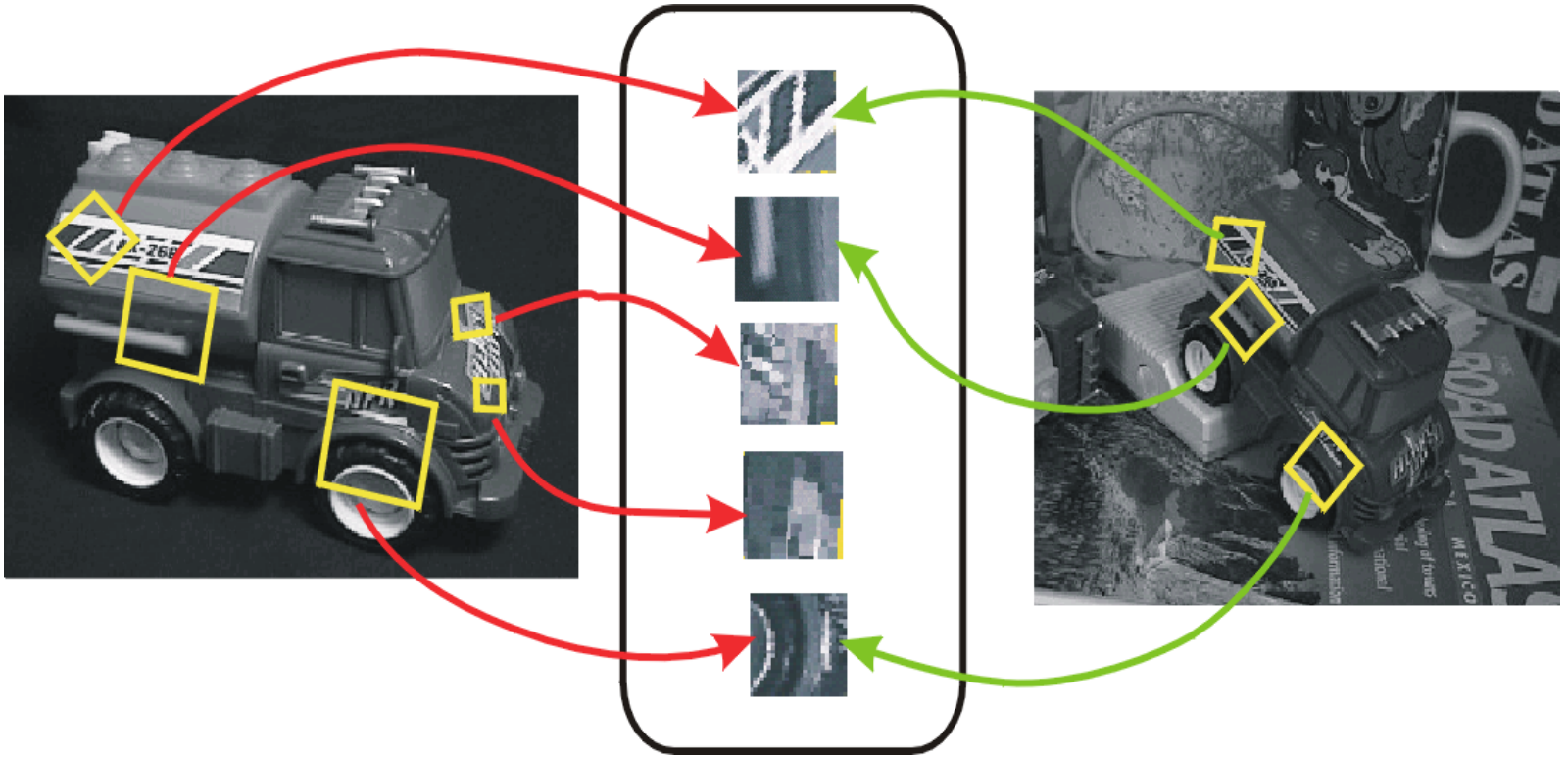
All points will be classified as **edges**, **not corners …**

(adapted from Kristen Grauman)

Invariant local features

Find features that are invariant to transformations

- geometric invariance: translation, rotation, scale
- photometric invariance: illumination, brightness, exposure, …



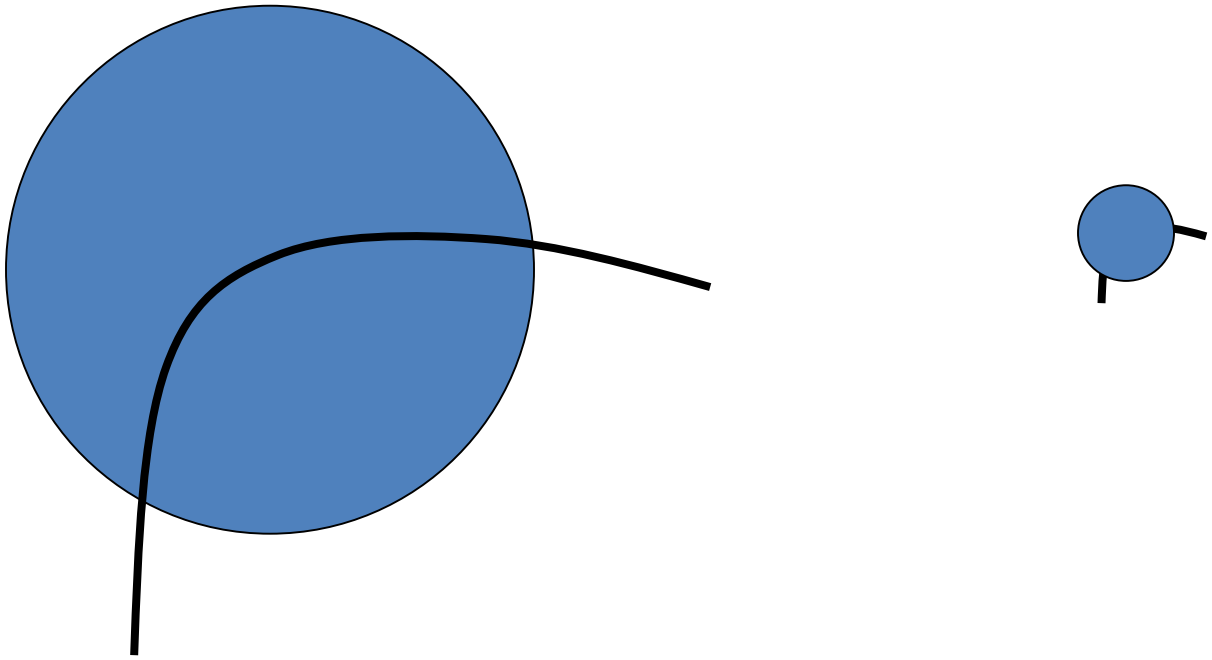
Feature descriptors

(in fact they are vectors of numbers)

David Lowe

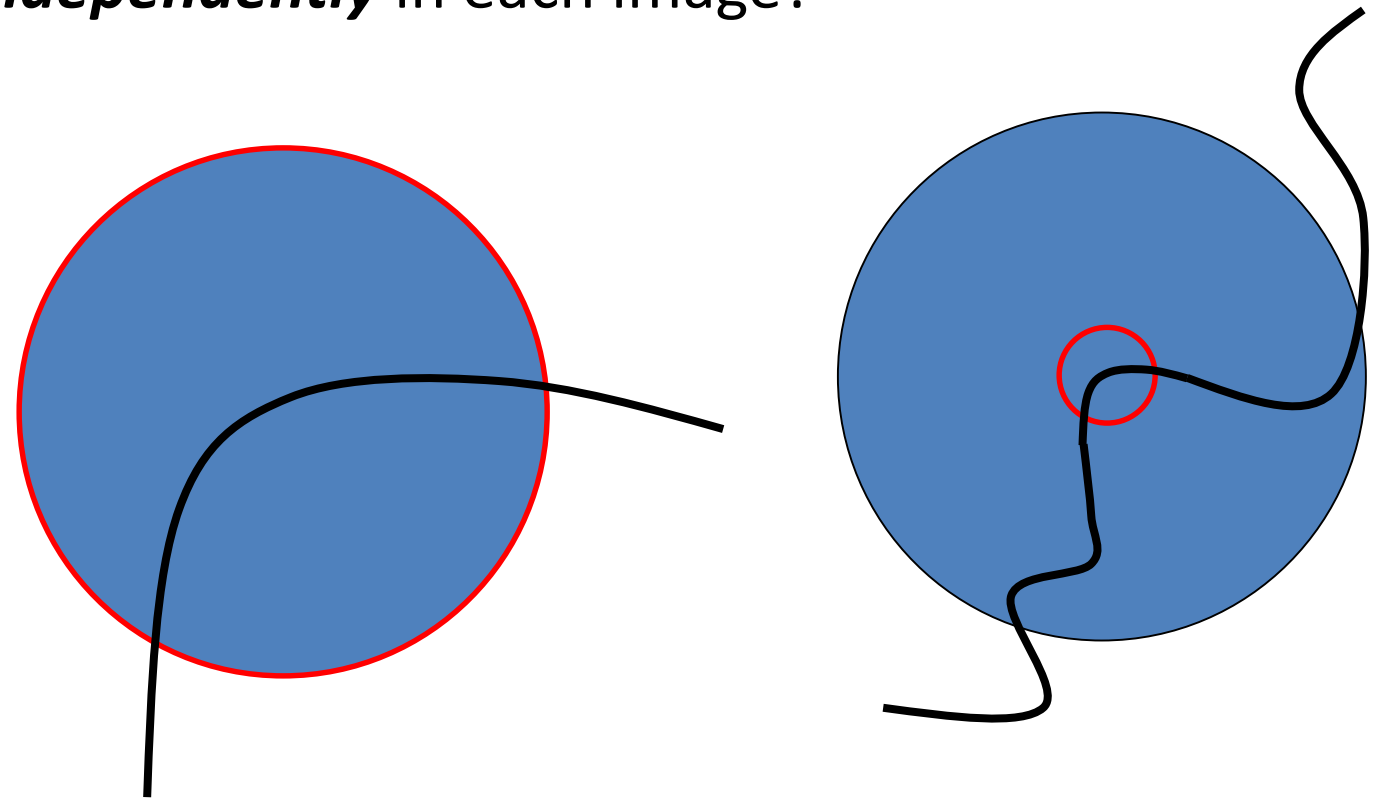
Scale invariant detection

- Consider regions (e.g. circles) of different sizes around a point
- Find regions of corresponding sizes that will look the same in both images?



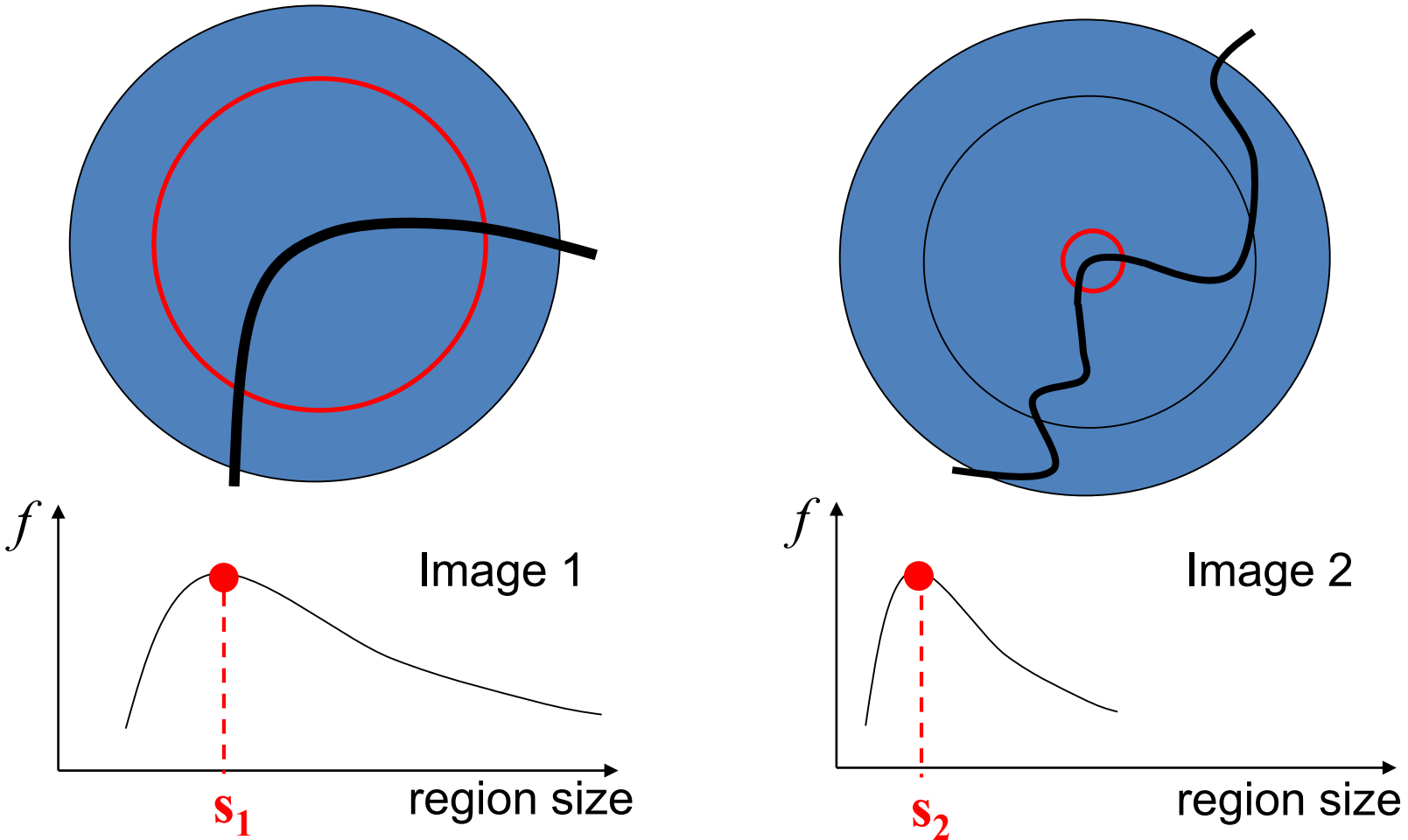
Scale invariant detection

- **Problem**:
  how do we choose
  corresponding regions
  independently in each image?



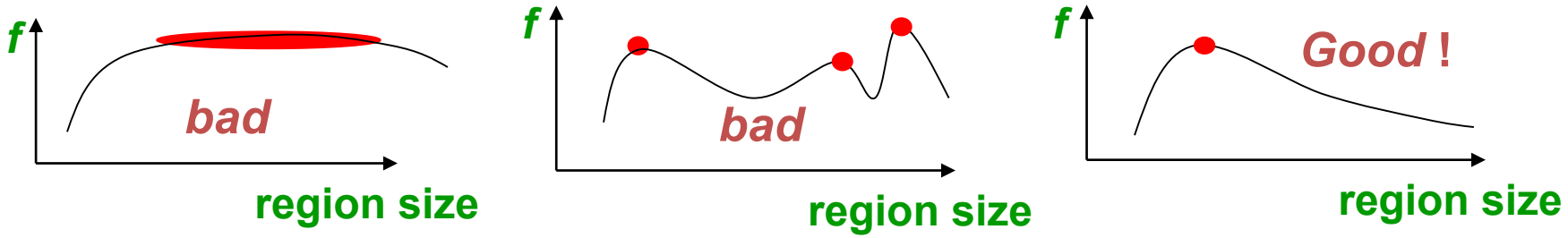
Scale invariant detection

- **Intuition**:
  find scale that gives local maxima of
  some signature function f, in both position and scale.



Scale invariant detection

- A "good" function for scale detection:
  has <u>one</u> stable sharp peak



- For usual images:
  a good function would be one which
  responds to contrast (sharp local intensity change)

Laplacian filter

- The Laplacian *L(x,y)* of an image with pixel intensity values *I(x,y)* is given by:

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

- This can be calculated using a convolution filter.
- Since the input image is represented as a set of discrete pixels, we have to find a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian.
Two commonly used small kernels are shown below.

0	−1	0
−1	4	−1
0	−1	0

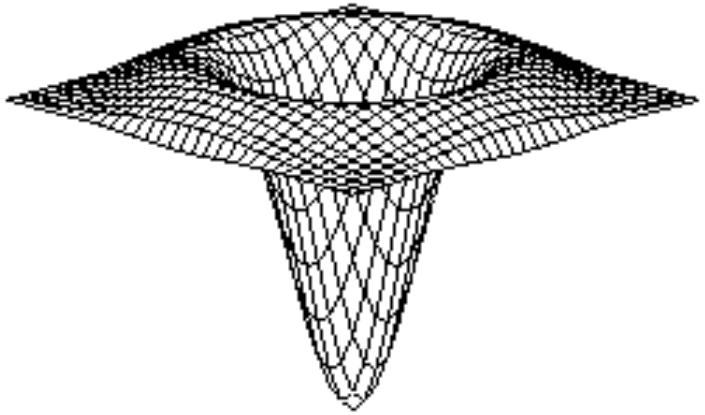
−1	−1	−1
−1	8	−1
−1	−1	−1

https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

Laplacian of Gaussian (LoG) filter

- The 2-D LoG function centered on zero and with Gaussian standard deviation σ has the form:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$



LoG function plot

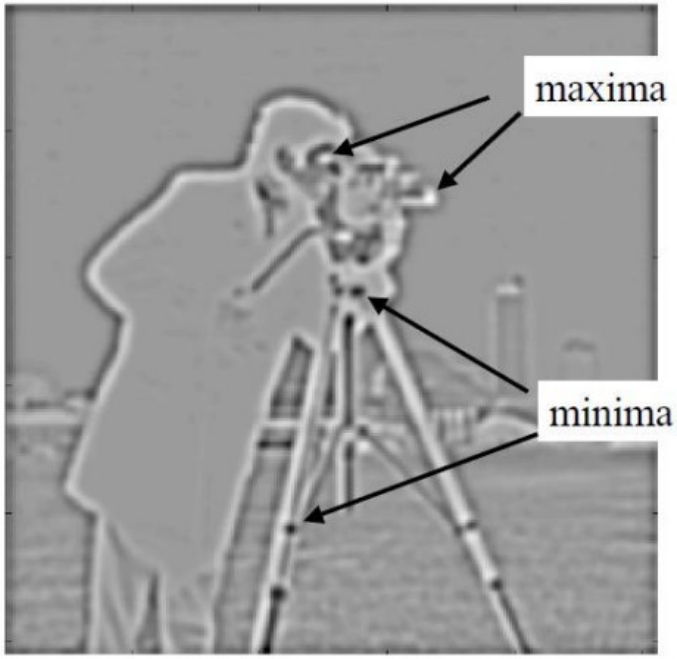
0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

A discrete kernel that approximates this function (for σ = 1.4)

adapted from William Hoff

Laplacian of Gaussian

- Laplacian of Gaussian - LoG ($\sigma = 2$)

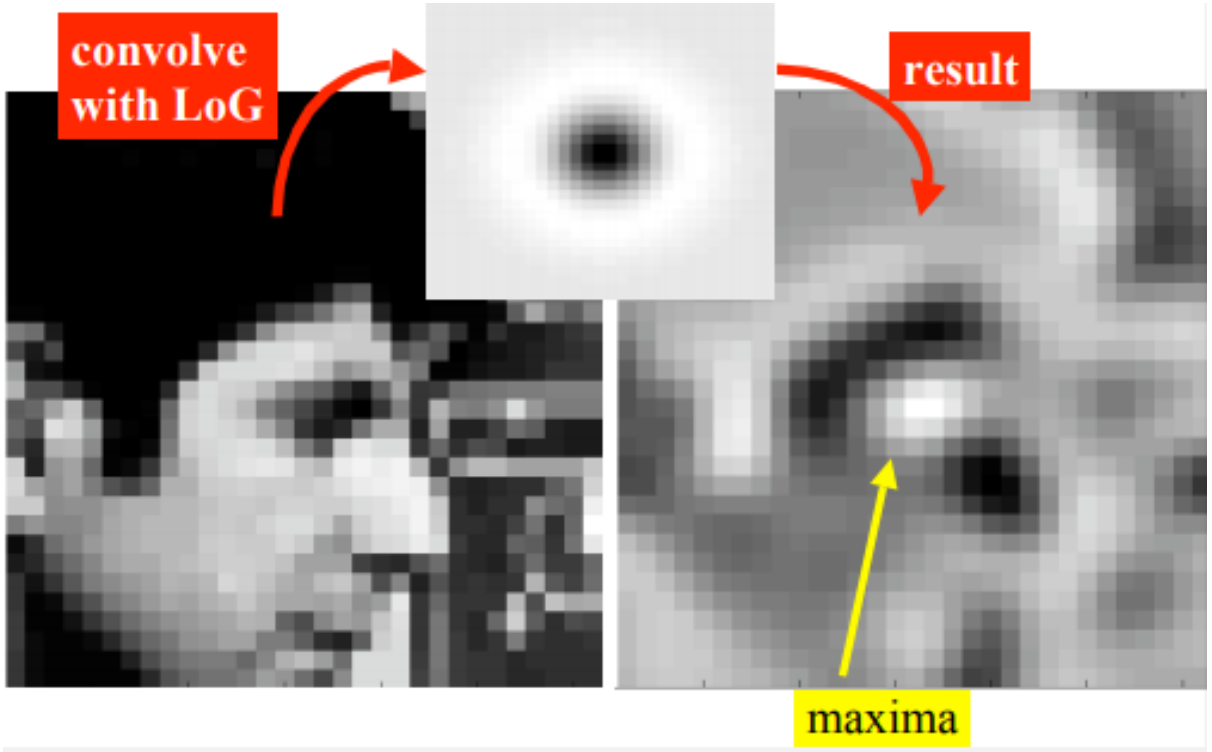


- LoG can be used as:
  - edge detector (zero crossings of the output image)
  - blob detector (maxima/minima of the output image)

adapted from William Hoff

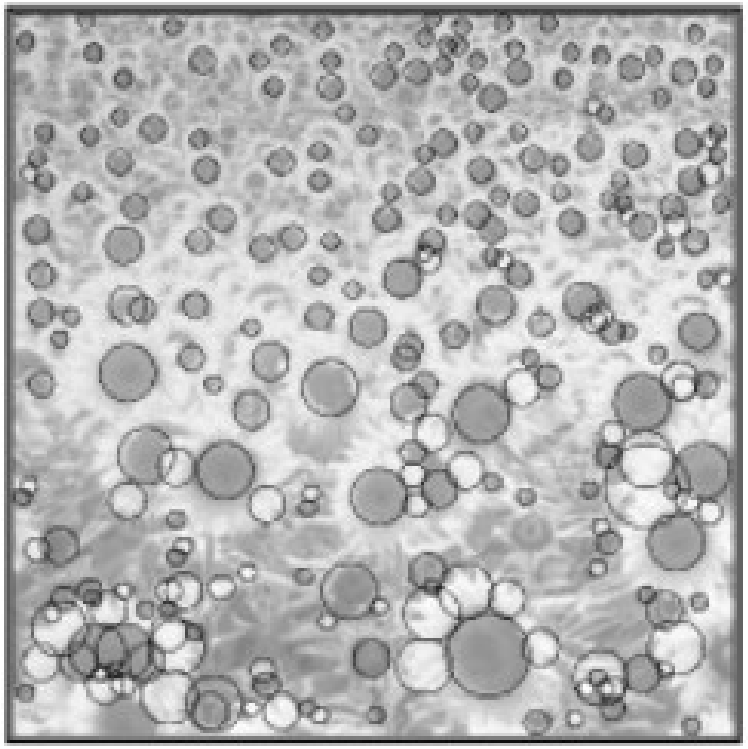
Laplacian of Gaussian

- LoG looks a bit like an "eye"
  and it responds maximally in the eye region !



adapted from Robert Collins (Penn State Univ., College of Engineering)

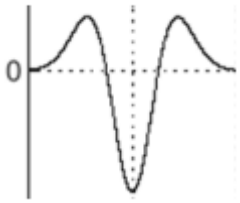
LoG for blob detection



Lindeberg: ``Feature detection with automatic scale selection".
International Journal of Computer Vision, vol 30, number 2, pp. 77-- 116, 1998

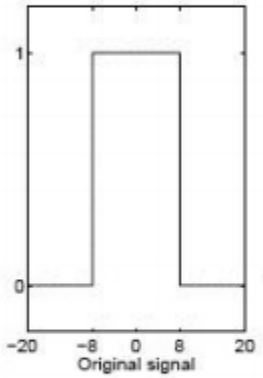
adapted from William Hoff

Automatic scale selection



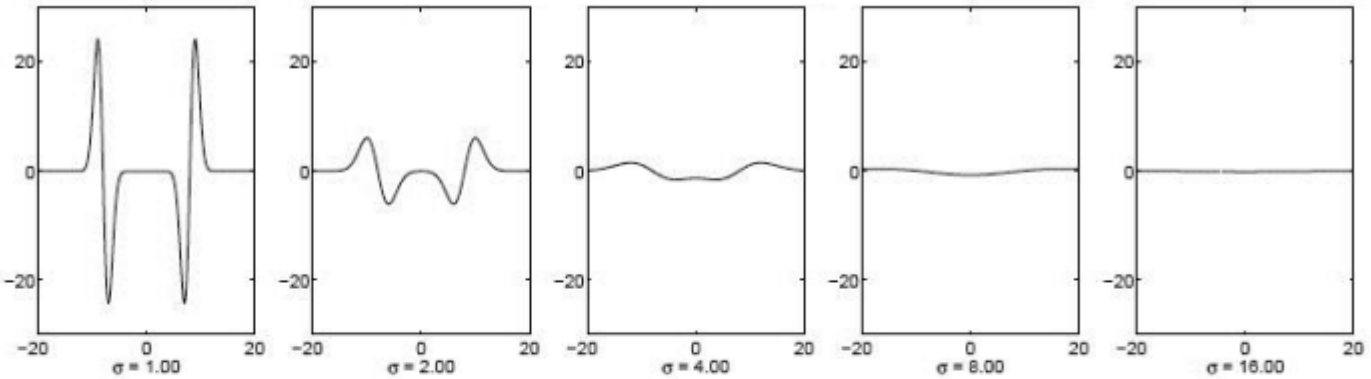
LoG filter

Original signal

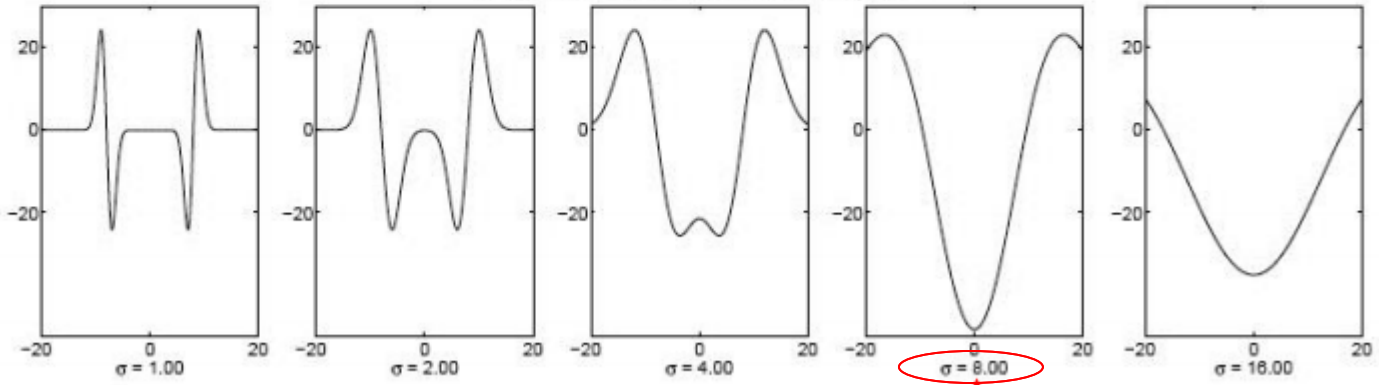


original signal
(radius=8)

Un-normalized Laplacian response



Scale-normalized Laplacian response



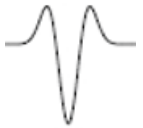
increasing σ

maximum

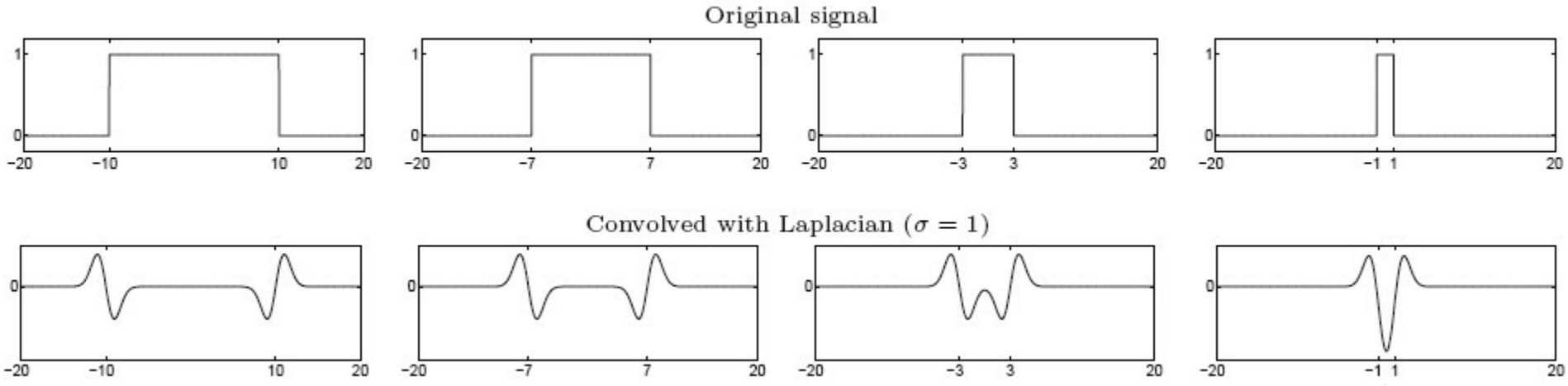
adapted from William Hoff

Scale invariance detection

- Edge = ripple
- Blob = superposition of two ripples



LoG filter
applied to all signals

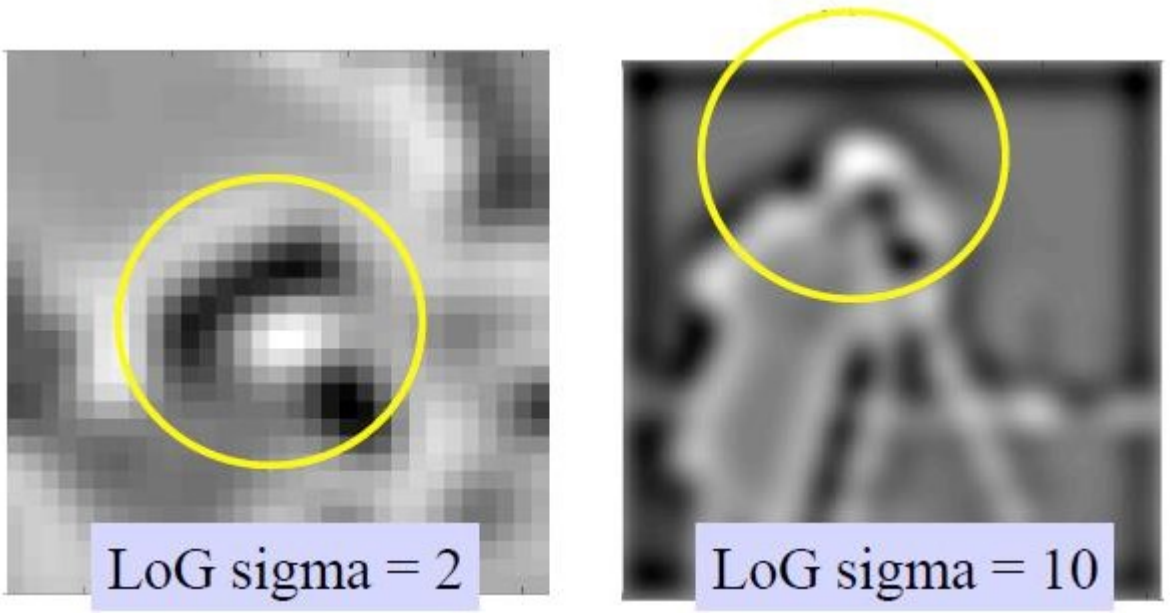


Spatial selection:
the magnitude of the Laplacian response
will achieve a maximum at the center of the blob,
provided the scale of the Laplacian is "matched"
to the scale of the blob

adapted from Lana Lazebnik

Automatic scale selection

- LoG filter extrema locates “blobs”
  - **maxima** = dark blobs on light background
  - **minima** = light blobs on dark background
- **Scale** of blob (size ; radius in pixels)
  is determined by the sigma parameter of the LoG filter.



adapted from William Hoff

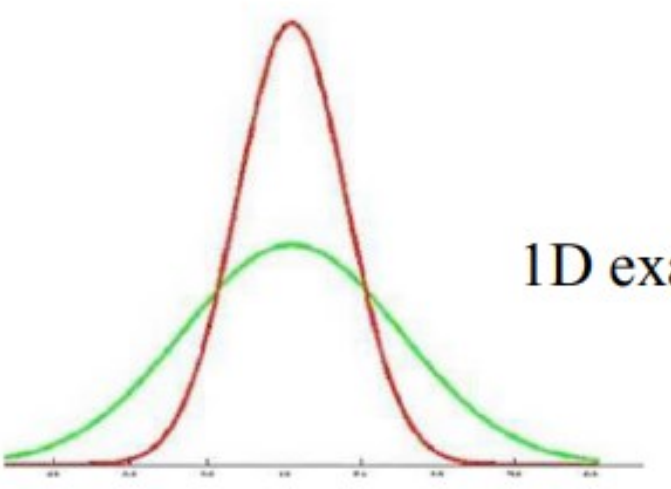
Approximating LoG with DoG

- LoG can be approximated by a difference of two Gaussians (DoG) at different scales

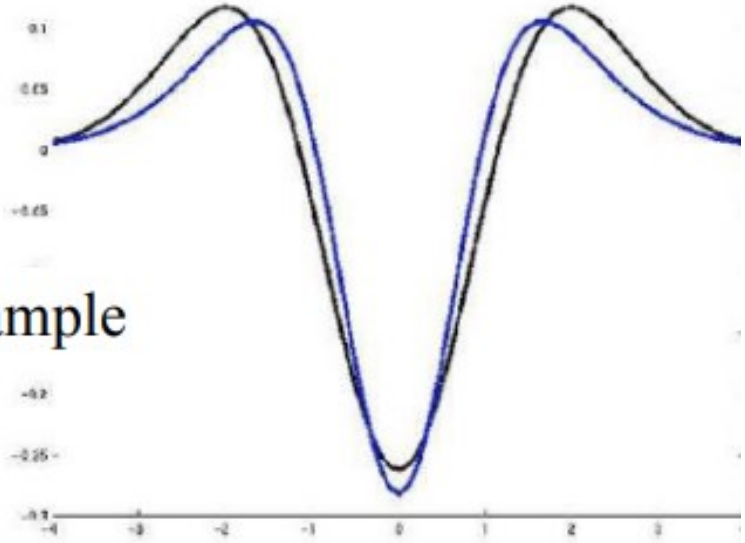
$$\nabla^2 G_\sigma \approx G_{\sigma_1} - G_{\sigma_2}$$

Best approximation when:

$\sigma_1 = \frac{\sigma}{\sqrt{2}}$, $\sigma_2 = \sqrt{2}\sigma$



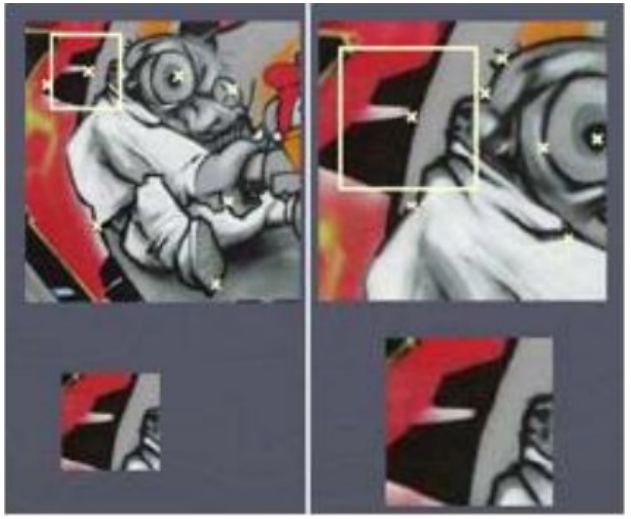
1D example



adapted from Robert Collins (Penn State Univ., College of Engineering)

SIFT – Scale Invariant Feature Transform

- D. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints”, IJCV, 2004
- Motivation:
  providing repeatable robust features for
  - Recognition
  - Panorama stitching
  - Tracking
  - etc.
- Characteristics:
  - Invariant to scaling and rotation.
  - Partly invariant to
    illumination and viewpoint changes.



- Very successful;
  experiments have shown
  it is one of the best approaches for feature matching.
- Widely used for recognizing objects from image databases.

SIFT - overview

- Detector
  - Create a scale space of images:
    - Construct a set of progressively Gaussian blurred images
    - Take differences to get a “difference of Gaussian” (DoG) pyramid (DoG is similar to LoG, *see previous slide*)
  - Find local extrema in this scale-space.
    Choose keypoints from the extrema.
    - Possibly apply a threshold to avoid select some of the extrema points.
- Descriptor
  - For each keypoint, in a 16x16 window,
    find histograms of gradient directions.
  - Create a feature descriptor (feature vector) out of these.

- http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
- http://www.vlfeat.org/api/sift.html
- http://en.wikipedia.org/wiki/Scale-invariant_feature_transform#Comparison_of_SIFT_features_with_other_local_features

SIFT - overview

- The SIFT detector uses as keypoints image structures which resemble "blobs"
- A SIFT keypoint is
  - a circular image region
  - with an orientation.
- It is described by a data structure which has many attributes, like:
  - the keypoint center coordinates (x,y)
  - its scale (the radius of the region)
  - its orientation (an angle)
  - the histogram of the image gradient around the keypoint
  - (...)
- By searching for blobs at multiple scales and positions, the SIFT detector is invariant (or, more accurately, covariant) to
  - translation,
  - rotations, and
  - re-scaling of the image.

For more information, see for example: http://www.vlfeat.org/api/sift.html

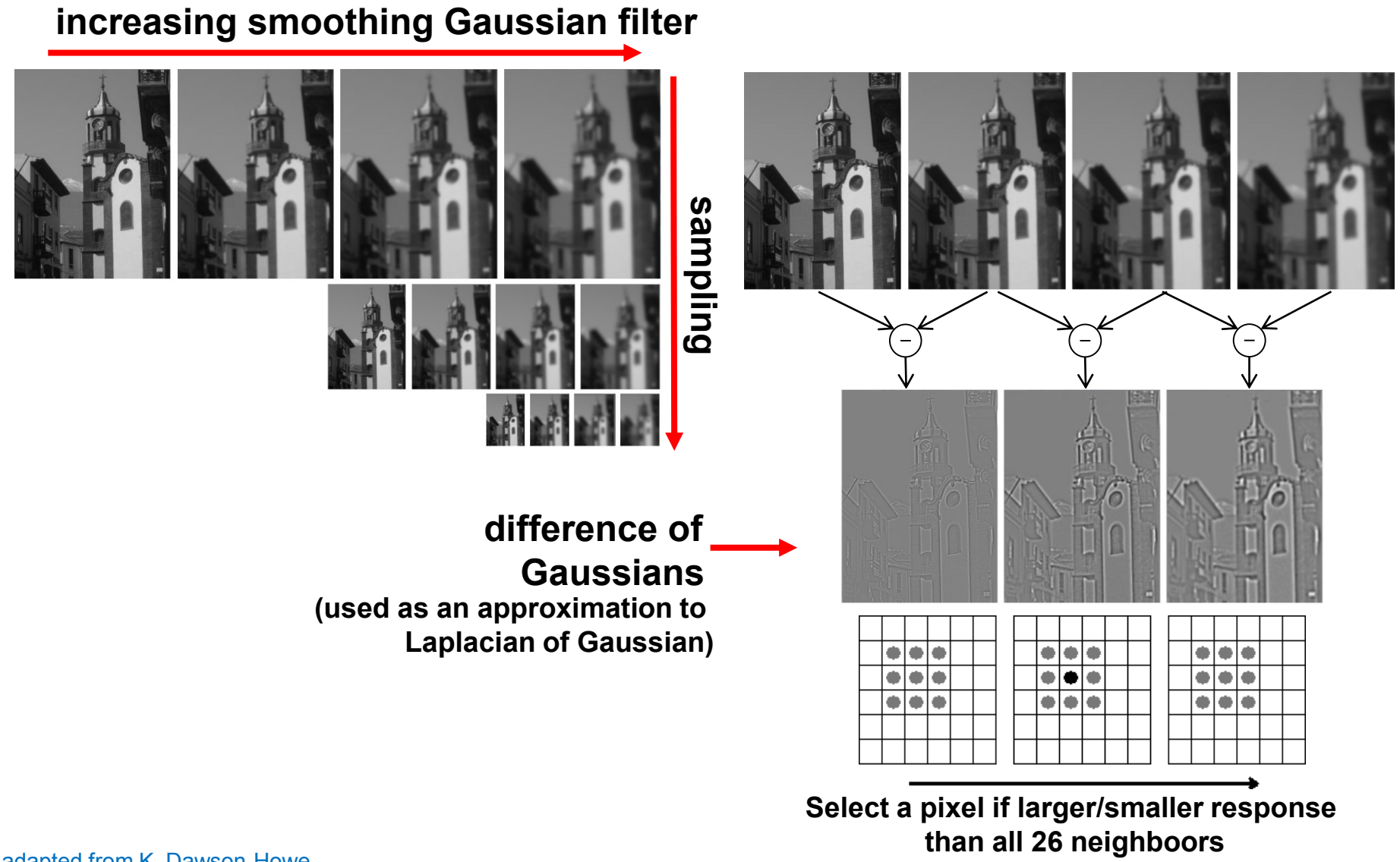
blob:
mainly a constant area but different from its surroundings

from the dictionary:
1. a globule of liquid; bubble.
2. a small lump, drop, splotch, or daub:
3. an object, especially a large one, having no distinct shape or definition

Invariance and covariance

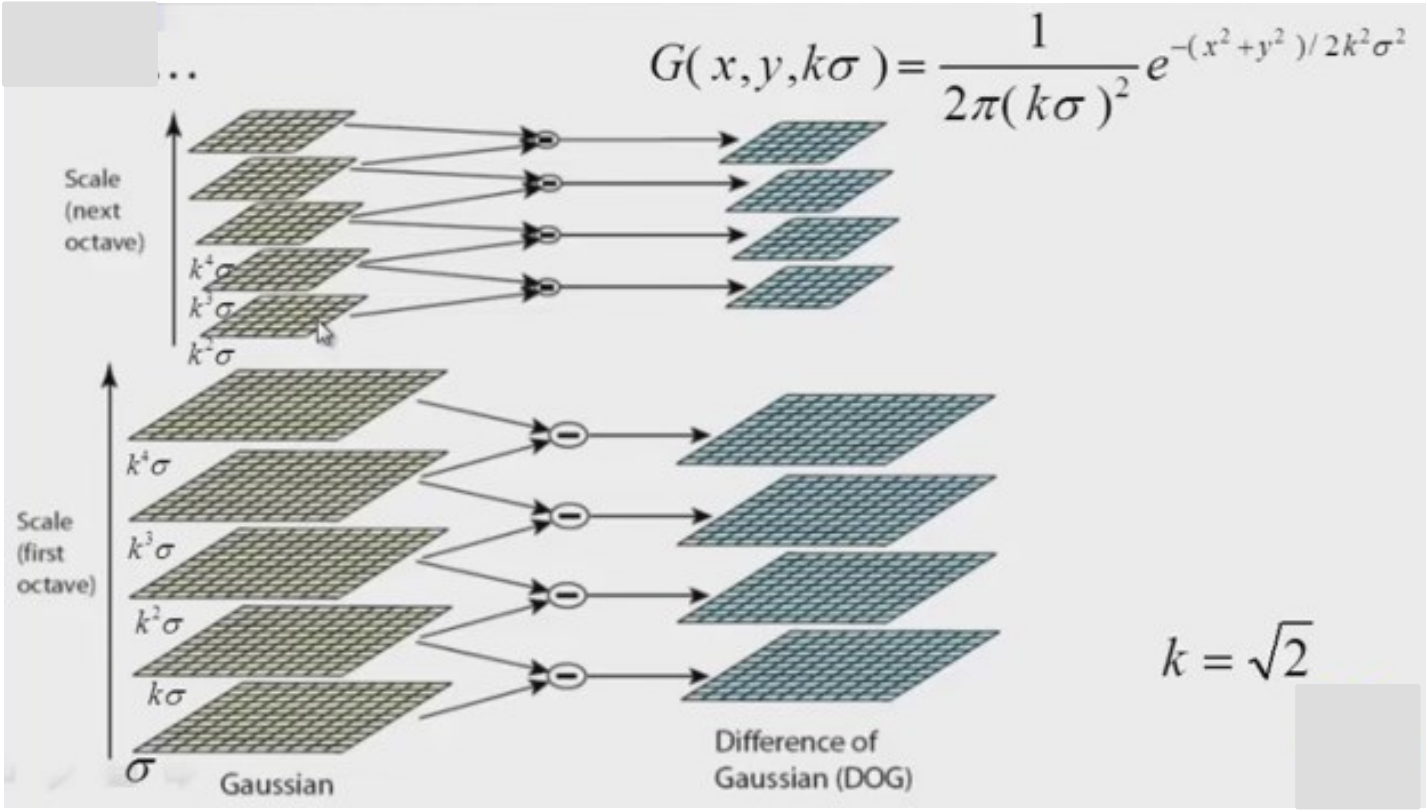
- We want feature point locations to be invariant to photometric transformations and covariant to geometric transformations
  - Invariance: when image is transformed the feature point locations do not change
    - features(transform(image)) = features(image)
  - Covariance: if we have two transformed versions of the same image, features should be detected in corresponding locations
    - features(transform(image)) = transform(features(image))
  - Covariant detection => invariant description

SIFT – scale space extrema detection

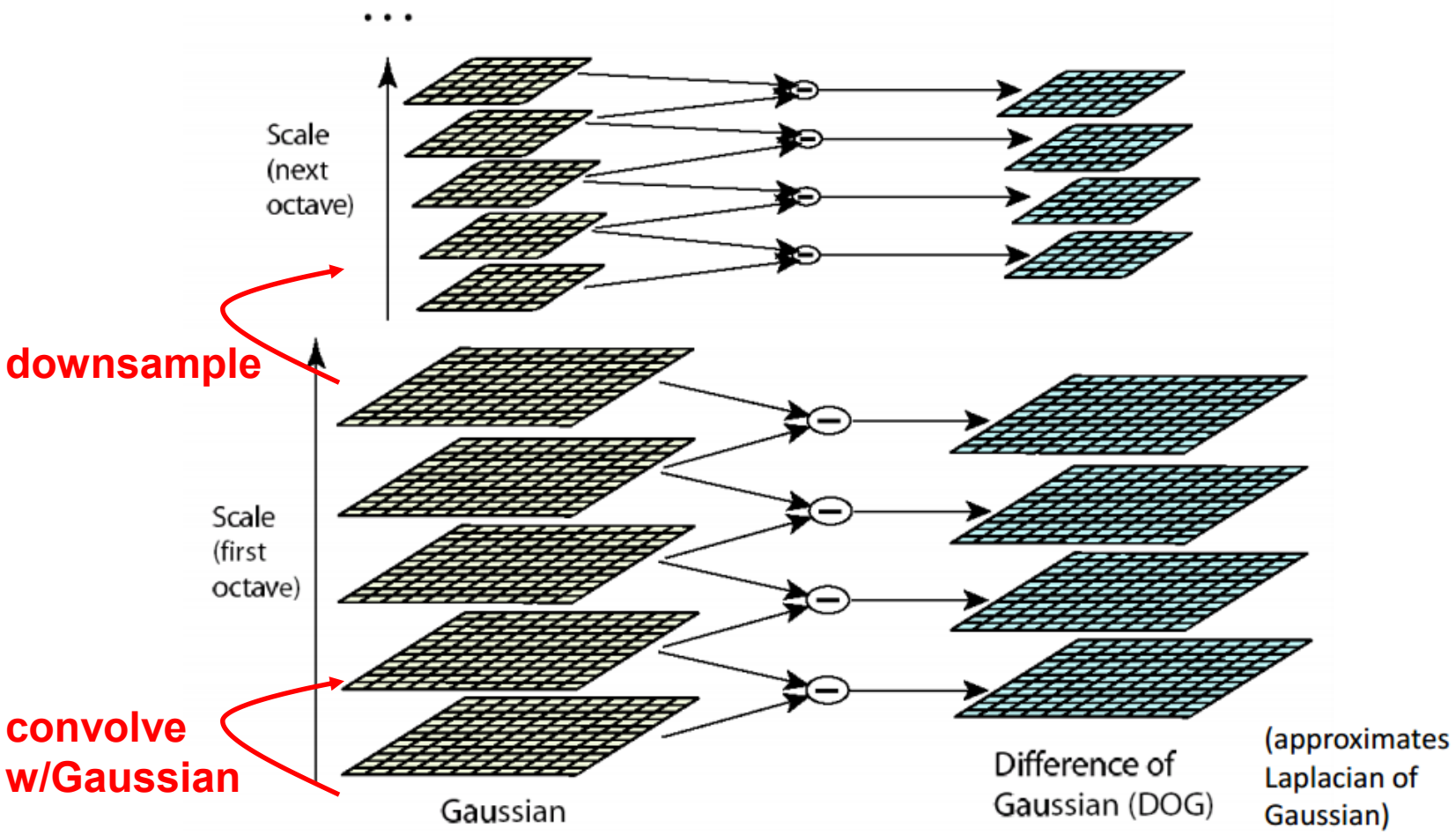


adapted from K. Dawson-Howe

SIFT - scale space images

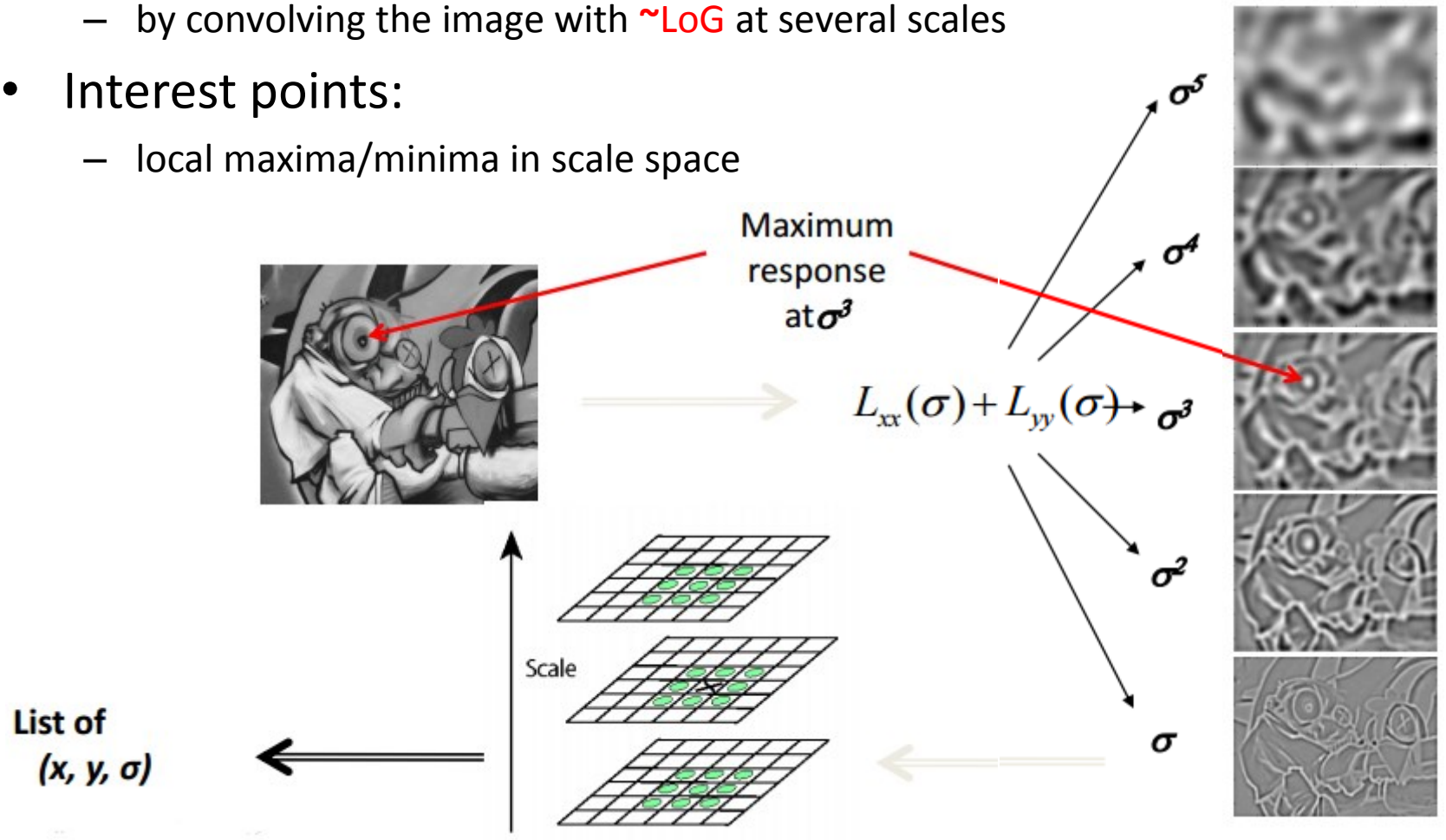


SIFT - scale space images



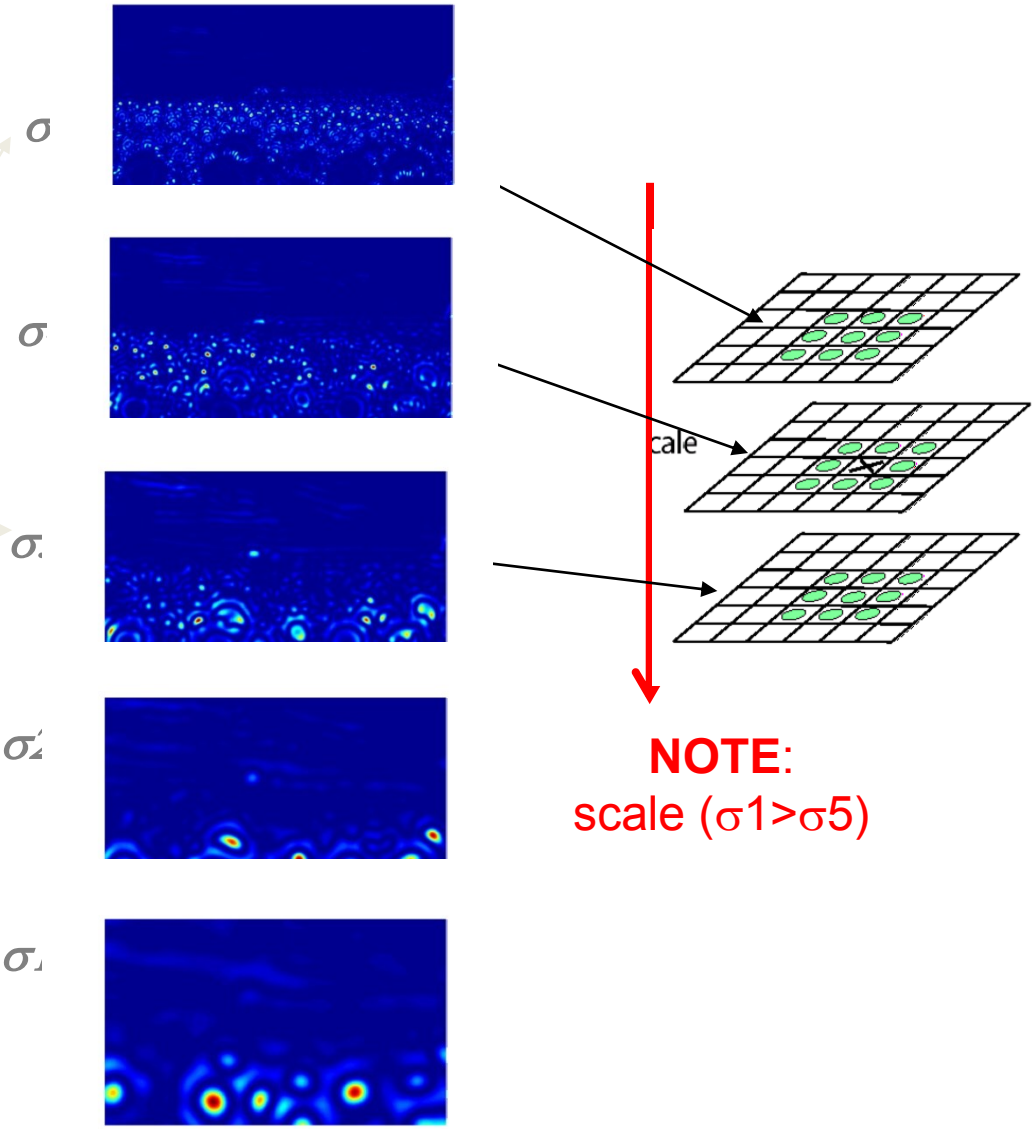
SIFT - automatic scale selection

- Find the characteristic scale of the blob
  - by convolving the image with ~LoG at several scales
- Interest points:
  - local maxima/minima in scale space

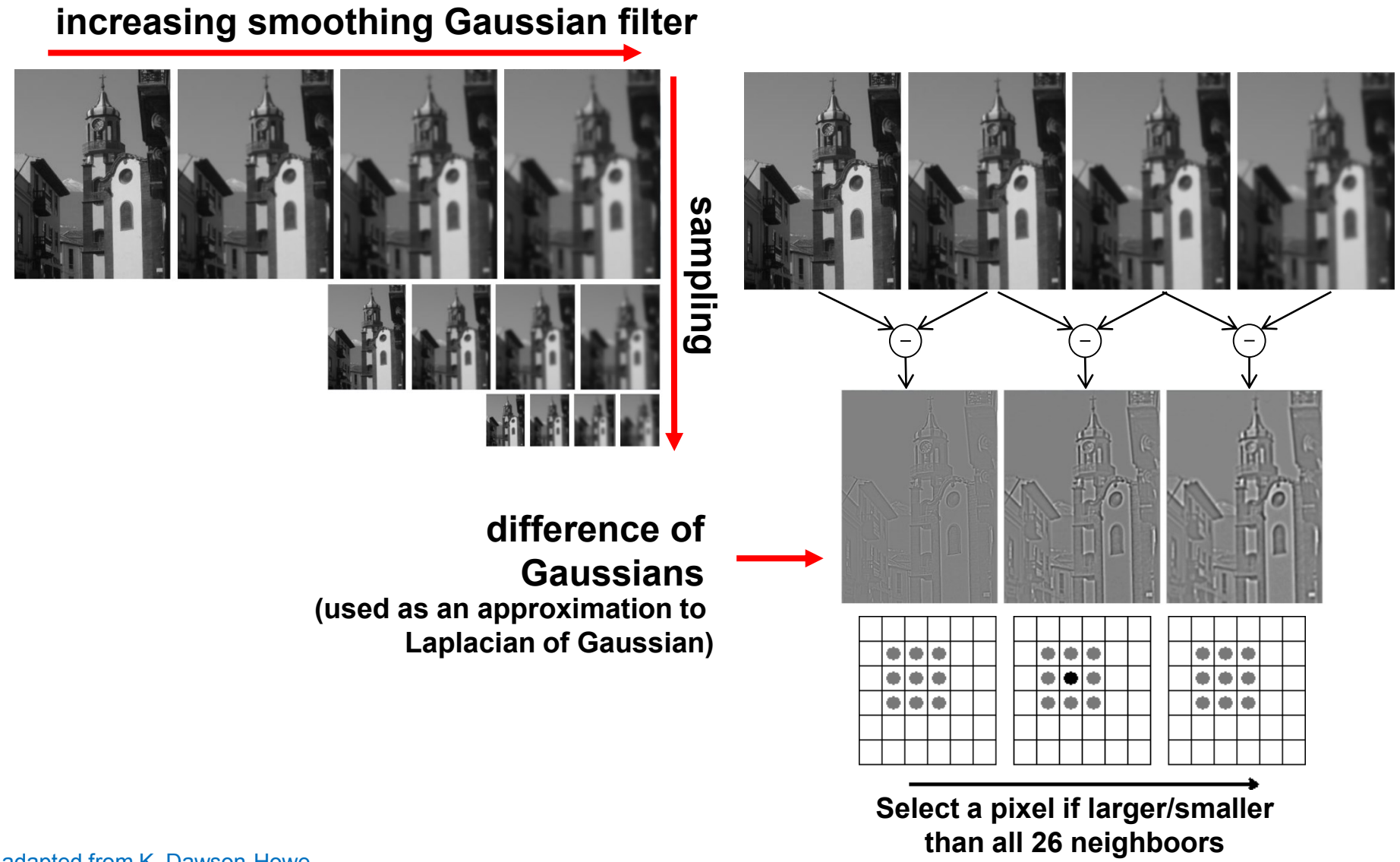


SIFT - scale invariant interest points

$$L_{xx}(\sigma)+L_{yy}(\sigma)$$



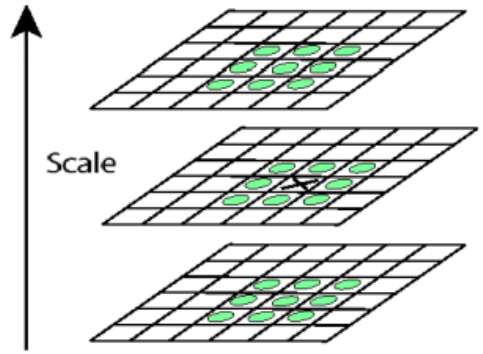
SIFT - scale space extrema detection



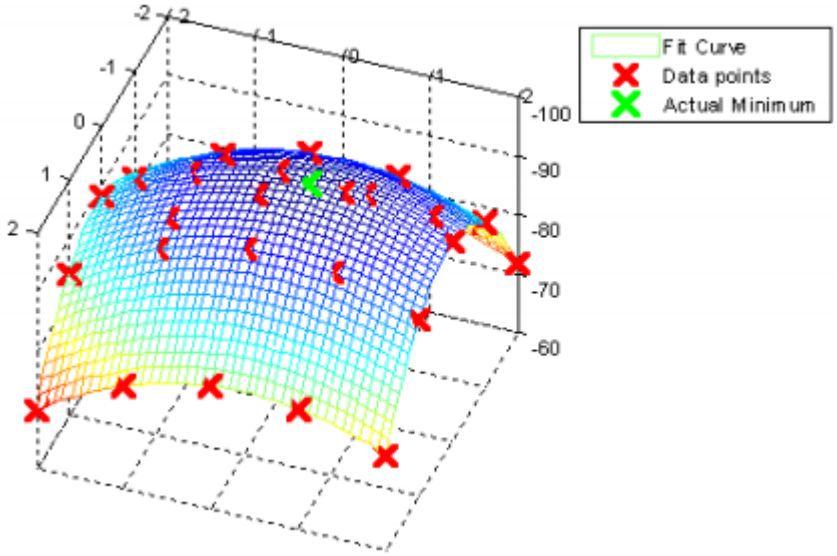
adapted from K. Dawson-Howe

Key point localization

- Detect maxima and minima of difference-of-Gaussians in scale space



- Fit a quadratic to surrounding values for sub-pixel and sub-scale interpolation

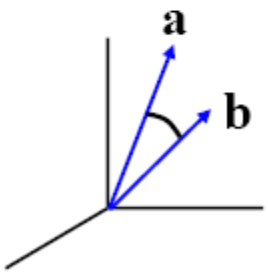


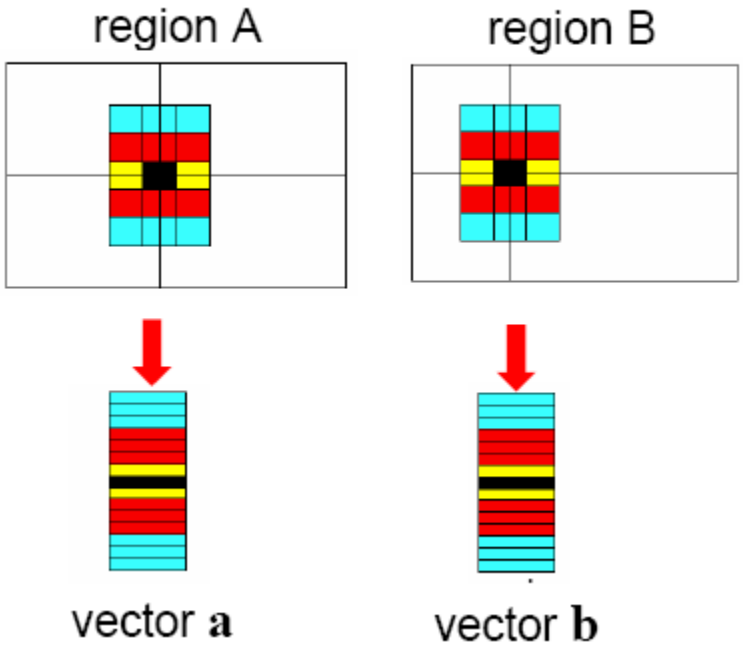
Local descriptors

- Simplest descriptor:
  list of intensities within a patch.
- What is this going to be invariant to?

Write regions as vectors

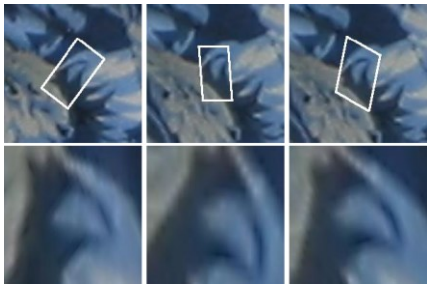
$$\mathrm{A} \rightarrow \mathbf{a}, \quad \mathrm{B} \rightarrow \mathbf{b}$$



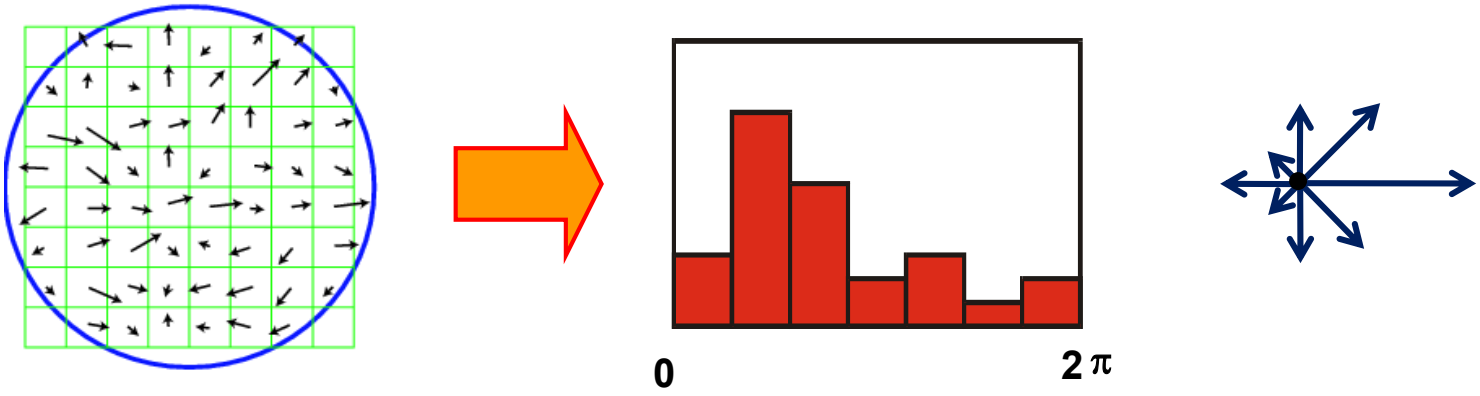


Local descriptors

- Disadvantage of patches as descriptors:
  - small shifts can affect matching score a lot



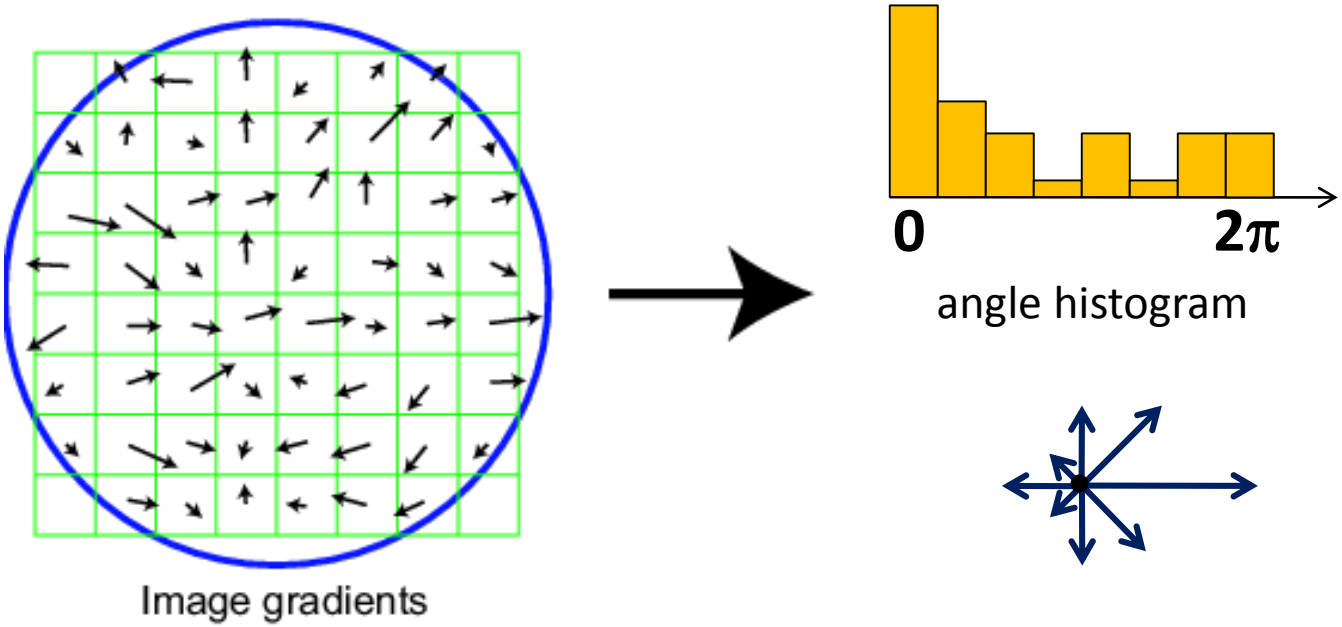
- Solution: histograms



SIFT descriptor

Basic idea:

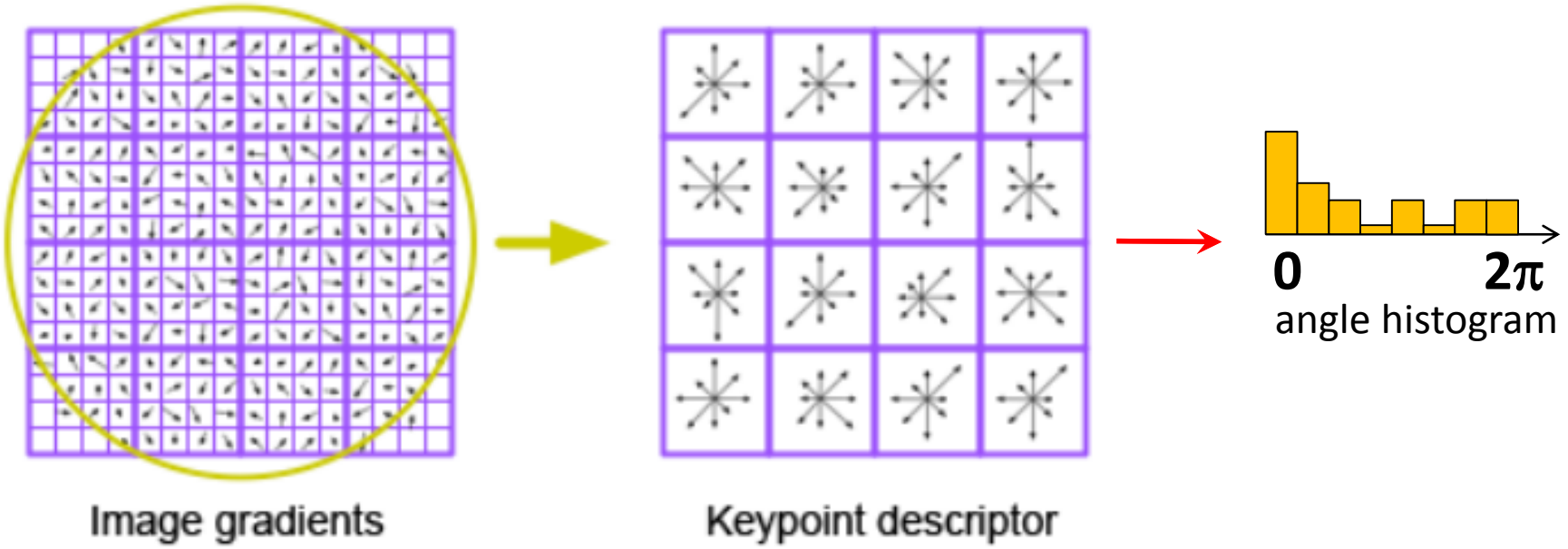
- Take 16x16 square window around detected feature
- Compute edge orientation for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations



Distinctive image features from scale-invariant keypoints. David G. Lowe. *IJCV* 60 (2), pp. 91-110, 2004.

SIFT descriptor

- Take 16x16 square window around detected feature
- Compute edge orientation for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Divide the 16x16 window into a 4x4 grid of cells
- Create histogram of surviving edge orientations
- Compute an orientation histogram for each cell
- Concatenate the histograms
- 16 cells * 8 orientations = 128 dimensional descriptor



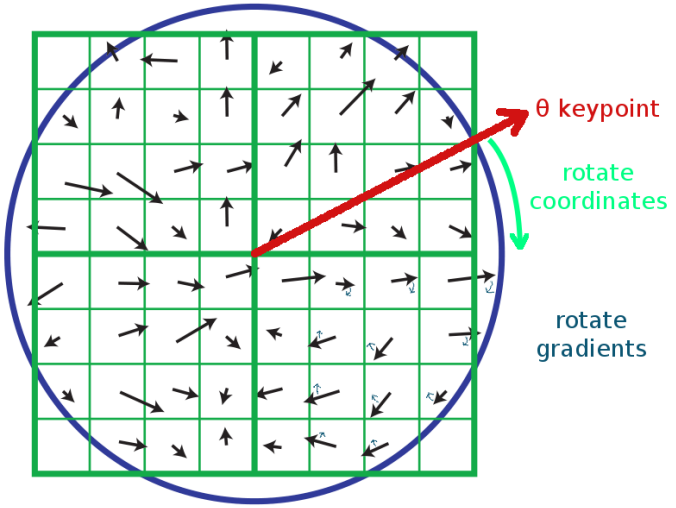
SIFT descriptor

Select canonical orientation

- Create histogram of local gradient directions computed at selected scale
- Assign canonical orientation at peak of smoothed histogram

SIFT descriptor

- Orientation adjustement
  - To become rotation invariant (/covariant),
    rotate the gradient directions and locations by (- θ);
    θ = keypoint orientation = peak of the gradient
    - in this way, rotation is cancelled out and
      gradients are expressed at locations relative to keypoint orientation



most prominent gradient

To handle situations where there may be
more than one dominant orientation around the interest point,
multiple peaks are accepted if the height of secondary peaks is
above 80 % of the height of the highest peak.
In the case of multiple peaks,
each peak is used for computing a new image descriptor
for the corresponding orientation estimate.

- Contrast normalization
  - To obtain contrast invariance, the SIFT descriptor is normalized to unit sum.
  - In this way, the weighted entries in the histogram will be invariant
    under local affine transformations of the image intensities around the interest point,
    which improves the robustness of the image descriptor under illumination variations.

SIFT - results

source: https://docs.opencv.org/3.3.0/da/df5/tutorial_py_sift_intro.html

- Scale represented by the size of the circle
- Notice the orientation associated to each feature

  (some interest points have more than one associated orientation – *see note in previous slide*)

SIFT summary

- Sparse, distinctive point features
- Sometimes unclear what features correspond to in the image
- **Translation** independent by using **local histogram**
- **Rotation** independent by **orientation adjustment**
- **Scale** independent by **extremal scale estimation**
- **Illumination** independent by **descriptor normalisation**
- Widely used
- Real-time implementation possible

Object recognition using SIFT

Note: objects can be recognized
even when partially occluded
or in the presence of clutter

Other feature point detectors/descriptors

- **Detectors**
  - Harris https://docs.opencv.org/4.x/dc/d0d/tutorial_py_features_harris.html
  - Shi-Tomasi https://docs.opencv.org/4.x/d4/d8c/tutorial_py_shi_tomasi.html
  - FAST (Features from Accelerated Segment Test) http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_fast/py_fast.html
  - MSER
  - CenSurE
- **Descriptors**
  - BRIEF (Binary Robust Independent Elementary Features) http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_brief/py_brief.html
  - Randomized trees
  - Ferns
  - PCA-SIFT
  - GLOH (Gradient Location and Orientation Histogram)
  - HOG (Histogram of Oriented Gradients)
- **Detectors & Descriptors**
  - SIFT (Scale-Invariant Feature Transform) http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
  - SURF (Speeded-Up Robust Features) http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html
  - ORB (Oriented FAST and Rotated BRIEF)
  - BRISK (Binary Robust Invariant Scalable Keypoints) http://docs.opencv.org/modules/features2d/doc/feature_detection_and_description.html

http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html

Feature matching

- The next task is to find correspondences between the keypoints found in both images.
- Brute-force matcher
  - Brute-Force (BF) matcher is simple
  - It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation
  - and the closest one is returned
- FLANN-based matcher
  - FLANN stands for **F**ast **L**ibrary for **A**pproximate **N**earest **N**eighbors
  - It implements a collection of algorithms optimized for fast nearest neighbor search in large datasets and for high dimensional features
  - It works more faster than BF matcher for large datasets

http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html

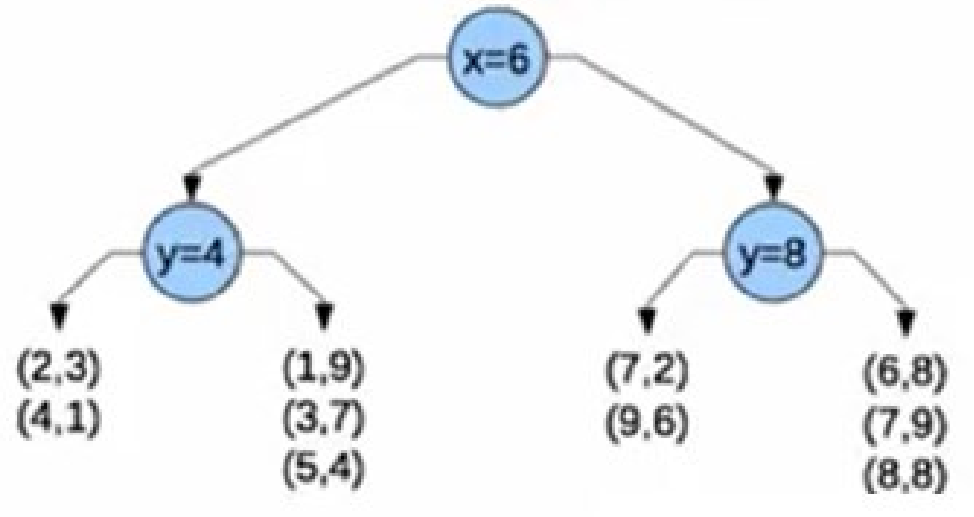
FLANN matcher

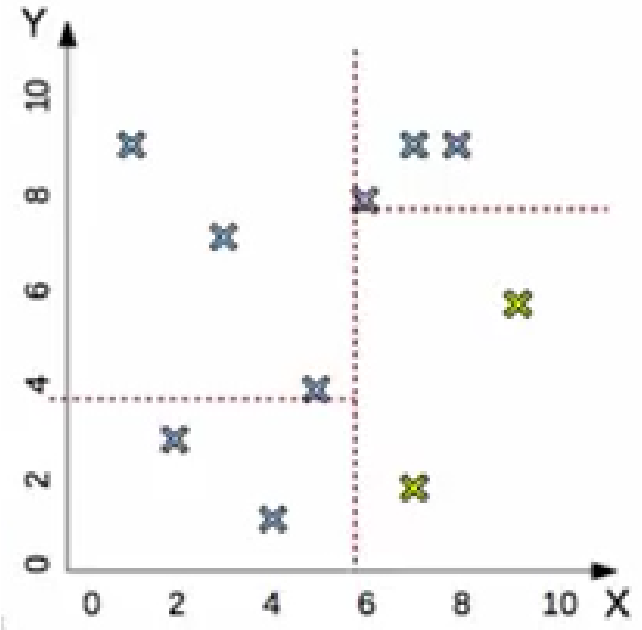
- In OpenCV, the FLANN matcher uses k-d trees, by default.
  Other methods can be used for matching.
- The extracted features are placed in k-d tree ,
  a data structure that can be searched efficiently.
  - a k-d tree (short for k-dimensional tree) is a space-partitioning data structure for organizing points in a k-dimensional space.
  - k-d trees are a useful data structure for several applications, such as searches involving a multidimensional search key (e.g. range searches and nearest neighbor searches).
  - k-d trees are a special case of binary space partitioning trees.
- Usually it is sufficient to look up all local feature-descriptors and match each one of them to his corresponding counterpart from the other image.
  - Note:
    two images from a scene
    don't necessarily have the same number of feature points (=> descriptors);
    some feature points may have no matching point

http://www.swarthmore.edu/NatSci/mzucker1/opencv-2.4.10-docs/modules/flann/doc/flann_fast_approximate_nearest_neighbor_search.html

k-d tree

- Building a k-d tree from "training"(*) data:
  - {(1,9), (2,3), (4,1), (3,7), (5,4), (6,8), (7,2), (8,8), (7,9), (9,6)}
  - pick random dimension (/attribute), find median, split data, repeat





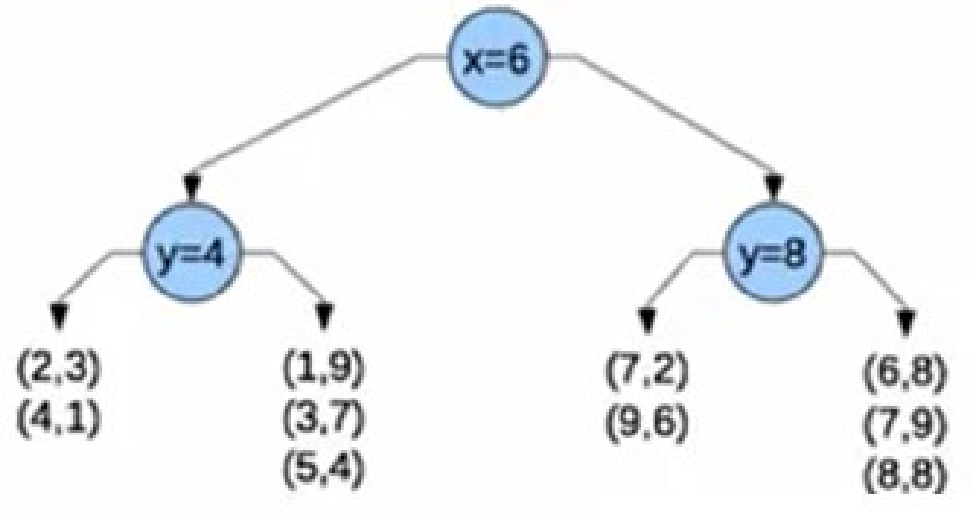
k-d = k-dimensional

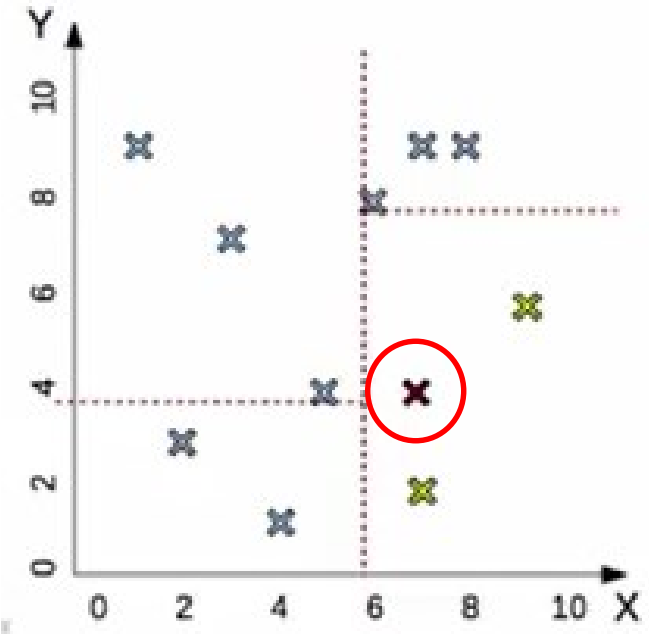
(*) - "Training" here means just "set of the descriptors, based on which kd-tree is build".

adapted from: "KD tree algorithm: how it works", Victor Lavrenko

k-d tree

- Find nearest neighbor for point (7,4):
  - find region containing (7,4)
  - compare to all points in region





- Notes:
  - it is an approximate technique because it is prone to mistakes (the closest neighbor may be in another region)
  - for an N-dimensional space, regions are hypercubes

adapted from: "KD tree algorithm: how it works", Victor Lavrenko

Distance measures

- In general, the distance $\mathbf{d_{ij}}$ between any two points in **n-dimensional space** may be calculated by the equation given by **Minkowski**:

$$d_{ij} = \left[\sum_{i=1}^{n} \left| x_{ik} - x_{jk} \right|^{p} \right]^{\frac{1}{p}}$$

  with **k** being the index of the coordinates,
  and **p** determining the type of distance.

- There are three special cases of the **Minkowski distance**:
  - **p = 1**: this distance measure is often called **city block distance,** or **Manhattan distance**.
  - **p = 1**, binary data: **Hamming distance**.
    - the Hamming distance defines the number of common "1" bits of two binary values.
  - **p = 2**: with **p** equalling 2 the Minkowski distance is reduced to the well-known **Euclidean distance**.

http://www.vias.org/tmdatanaleng/cc_distance_meas.html

Distance measures

- Other distance measures
  - **Cosine similarity:**
    Given two vectors of attributes, A and B,
    the cosine similarity, cos(θ),
    is represented using a dot product and magnitude as

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

http://www.vias.org/tmdatanaleng/cc_distance_meas.html

Distance measures

- Euclidean distance
  - Is the "ordinary" distance between two points in Euclidean space, i.e., the length of the line segment connecting them.
  - In Euclidean n-space: $\mathrm{d}(\mathbf{p}, \mathbf{q}) = \mathrm{d}(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$
- Hamming distance
  - In information theory, the Hamming distance between two "strings" of equal length is the number of positions at which the corresponding symbols are different.
  - The Hamming distance between:
    - "**John**" and "**Joao**" is 2.
    - **101110** and **100100** is 2.
    - **1234** and **2243** is 3.
  - Hamming distance can be seen as Manhattan distance between bit vectors.
  - Binary descriptors (ORB, BRISK, ...) are matched using the Hamming distance.

J. Ventura, T. Hollerer, "Fast and scalable keypoint recognition and image retrieval using binary codes", *IEEE Workshop onApplications of Computer Vision (WACV), 2011*

Distance measures - Mahalanobis

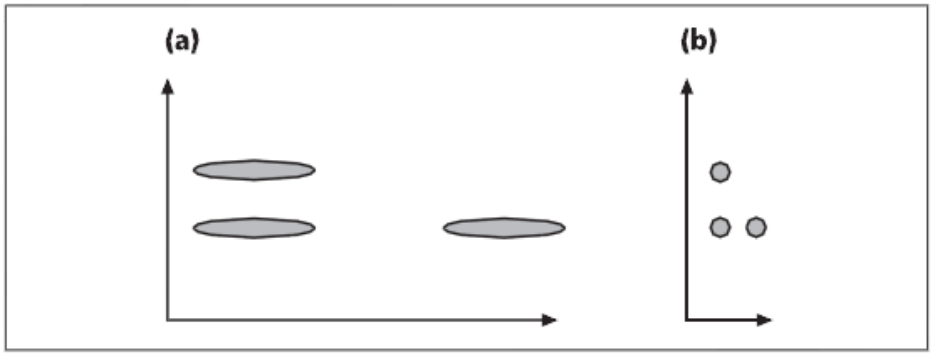
- The various forms of the Minkowski distance do not account for different metrics of the individual coordinates.
- If the coordinates span different ranges, the coordinate with the largest range will dominate the results.
- You therefore have to scale the data before calculating the distances.
- Furthermore, any correlations between variables (coordinates) will also distort the distances.
- In order to overcome these drawbacks, one should calculate the **Mahalanobis distance** which allows for correlation and different scalings.
- The Mahalanobis distance is related to the Euclidean distance. It can easily be calculated by including the inverse covariance matrix (C^{-1}) into the distance computations:

$$d_{ij} = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T C^{-1} (\mathbf{x}_i - \mathbf{x}_j)}$$

Distance measures - Mahalanobis

Mahalanobis distance:

- A distance measure that takes into account the variance of the data.
- In particular,
  distance from the mean along a direction where there is little variance has a large weight and
  distance from the mean along a direction where there is a large variance has little weight.
- If the covariance is the identity matrix (identical variance),
  then this measure is identical to the Euclidean distance measure.



The Mahalanobis computation allows us to reinterpret the data's covariance
as a "stretch" of the space:

- the vertical distance between raw data sets is less than the horizontal distance;
- after the space is normalized for variance,
  the horizontal distance between data sets is less than the vertical distance

Distance measures

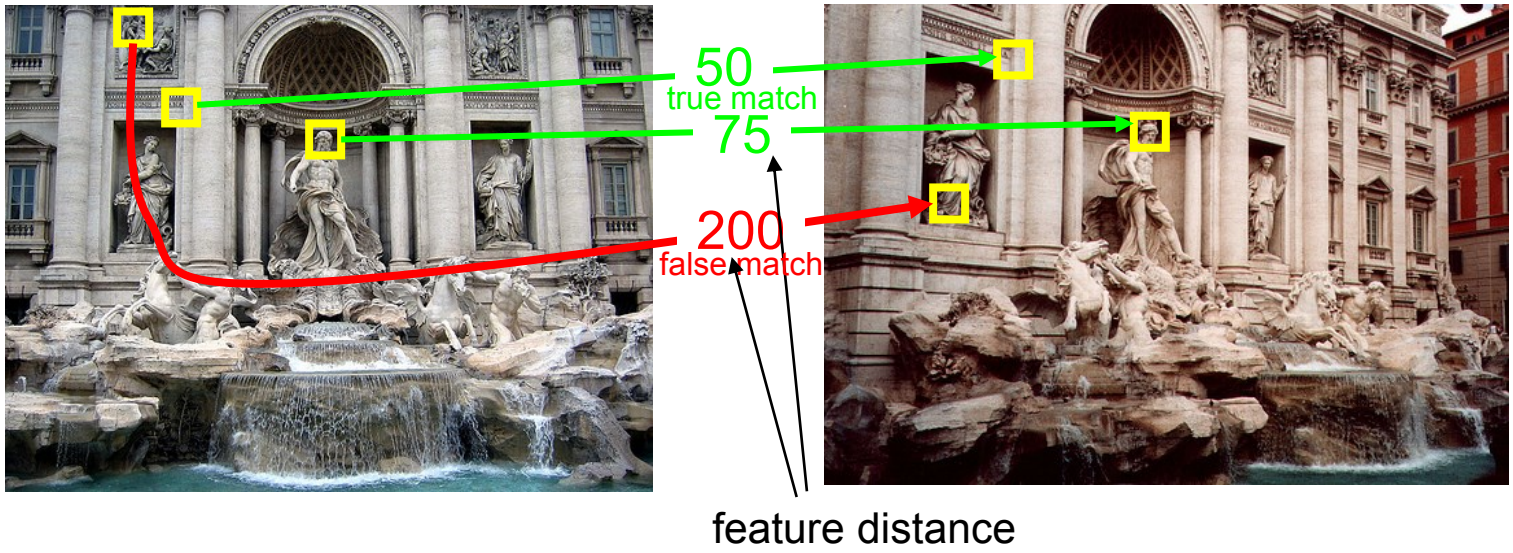
- Another distance measure, which is rather a measure for the **similarity** between two objects, has been proposed by **Jaccard** (**Jaccard coefficient;** it is also called **Tanimoto coefficient**):

$$T(x,y) = \frac{(x \cdot y)}{\|x\|^2 + \|y\|^2 - (x \cdot y)}$$

with **(x.y)** being the inner product of the two vectors, **x** and **y**.

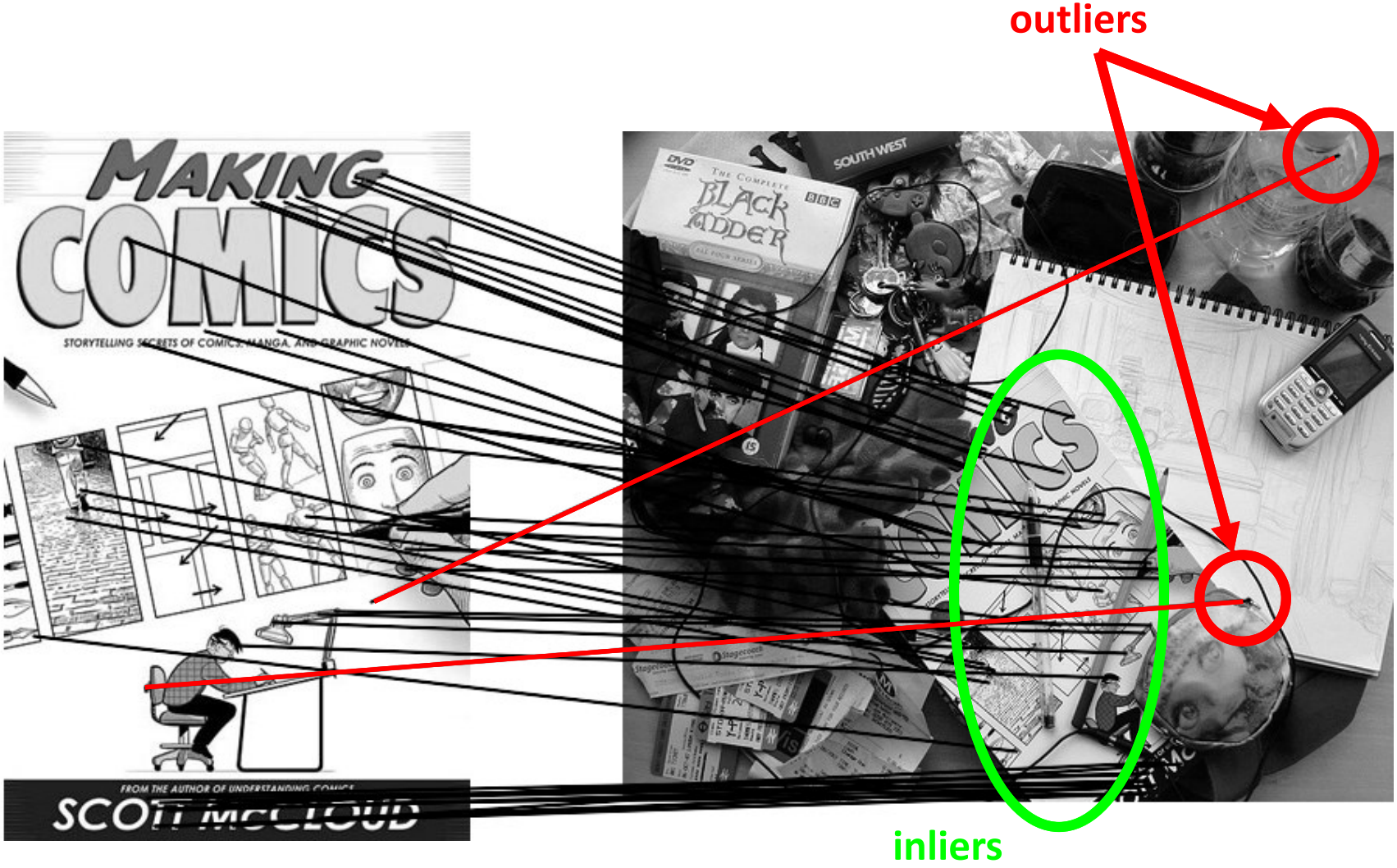
- Note that the Jaccard coefficient equals 1.0 for objects with zero distance.

Feature matching



- **Eliminate bad matches**: throw out features with distance > threshold
- And, eliminate **non consistent** matches …

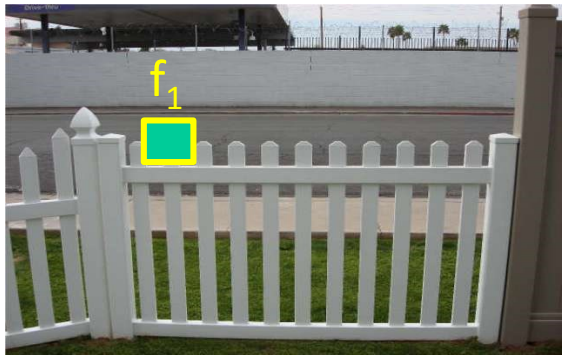
Outliers


outliers
MAKING COMICS
STORYTELLING SECRETS OF COMICS, MANGA, AND GRAPHIC NOVELS
FROM THE AUTHOR OF UNDERSTANDING COMICS
SCOTT MCCLOUD
THE COMPLETE BLACK ADDER
BBC
DVD
SOUTH WEST
inliers

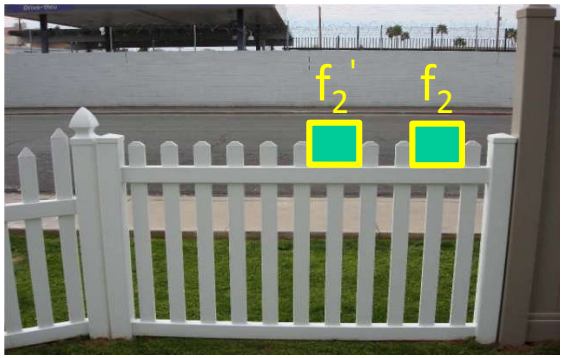
Consistency checks – distance ratio

Distance ratio test

- To filter the matches,
  Lowe proposed to use a **distance ratio test** to try to eliminate false matches.
- The distance ratio between the two nearest matches of a considered keypoint is computed and it is a good match when this ratio is below a threshold.
  Indeed, this ratio allows helping to discriminate between ambiguous matches (distance ratio between the two nearest neighbors is close to one) and well discriminated matches.
- **Distance ratio = distance(f_1 , f_2) / distance(f_1 , f_2')**
  - the best match to f_1 in **Image**$_1$ is f_2 in **Image**$_2$
  - the 2nd best match to f_1 in **Image**$_1$ is f_2' is in **Image**$_2$
  - the ratio is ~1 for ambiguous matches



Image$_1$

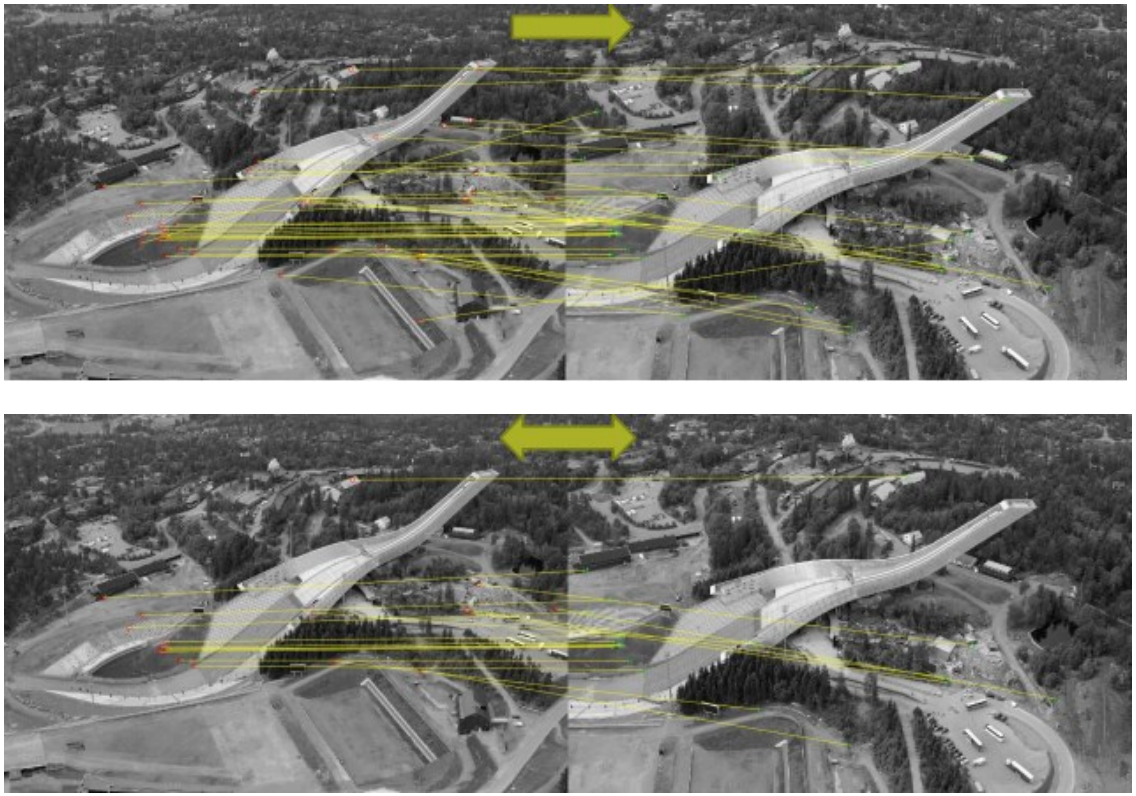


Image$_2$

Consistency checks – cross check

Cross check test

- good match (f_1,f_2) if
  feature f_2 is the best match for f_1 in $Image_2$ and
  feature f_1 is the best match for f_2 in $Image_1$

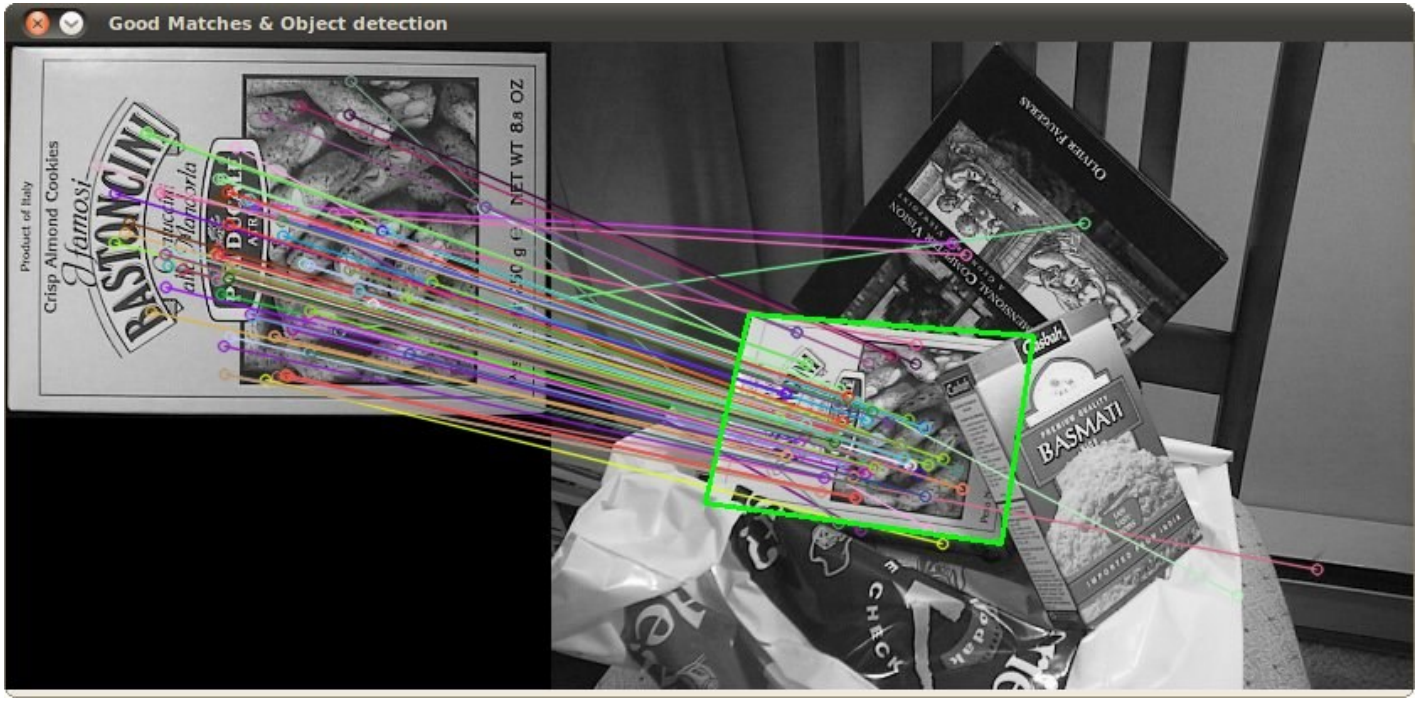


images by
T. Haavardsholm

Consistency checks – geometric test

Geometric test

- eliminate matches that do not fit to a geometric model,
  e.g. RANSAC homography for planar objects

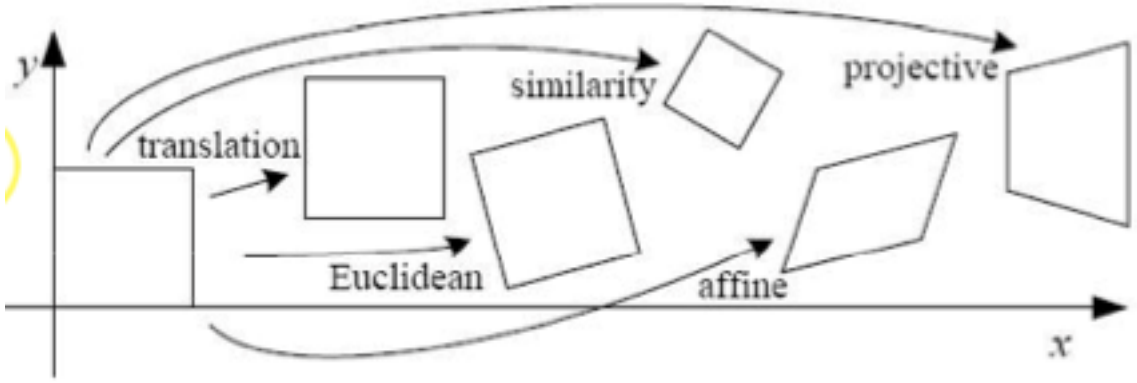


Invariant local features

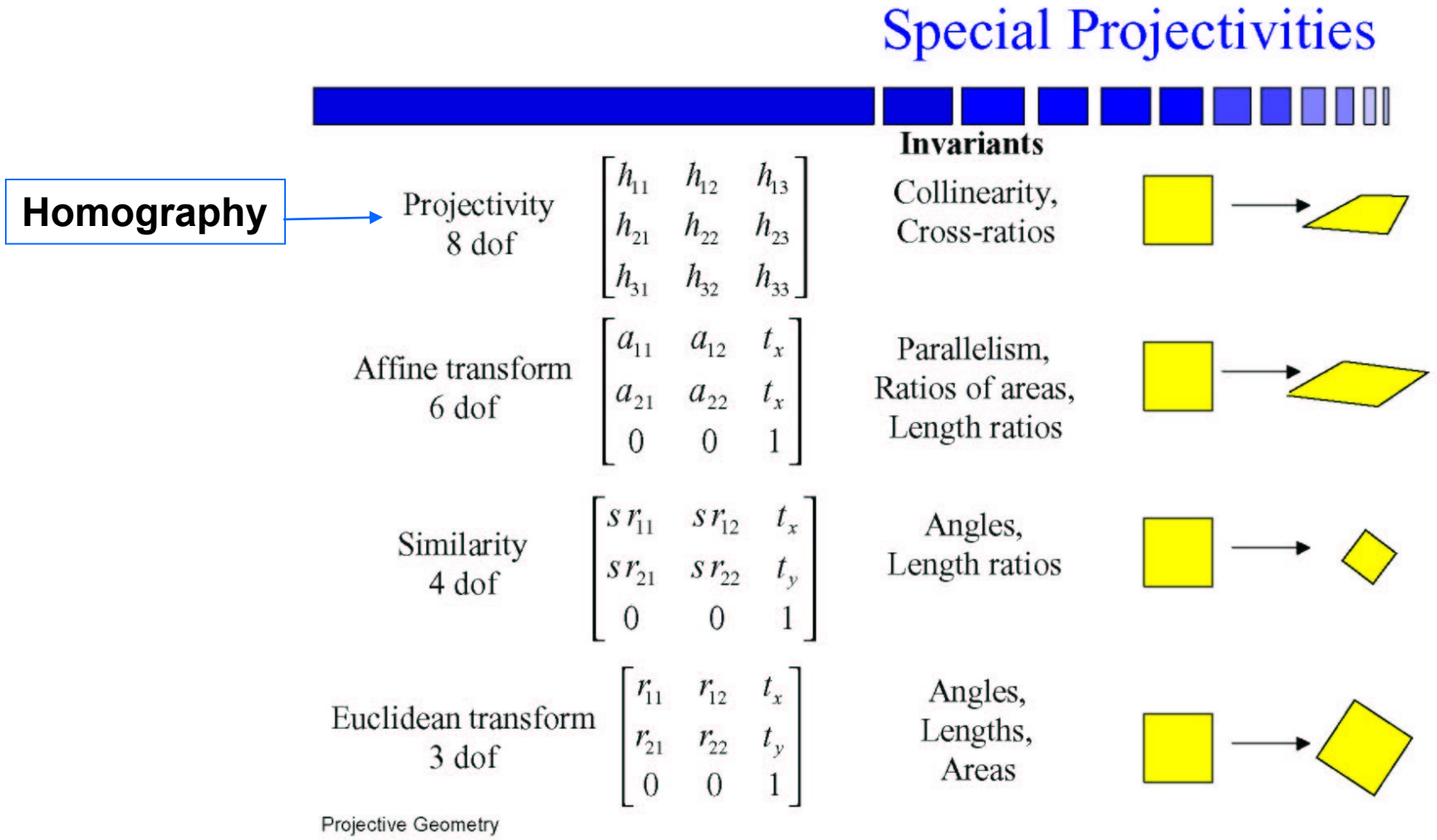
Geometric transformations

- Translation
- Euclidean (translation + rotation)
- Similarity (translation + rotation + scale)
- Affine transformations
- Projective transformations (homography)

Only holds for planar patches

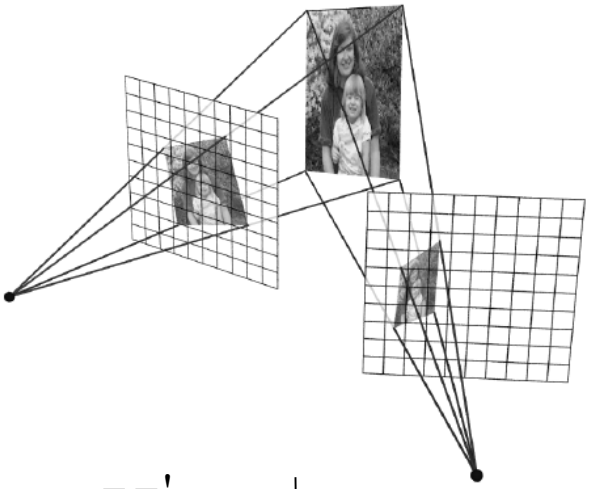


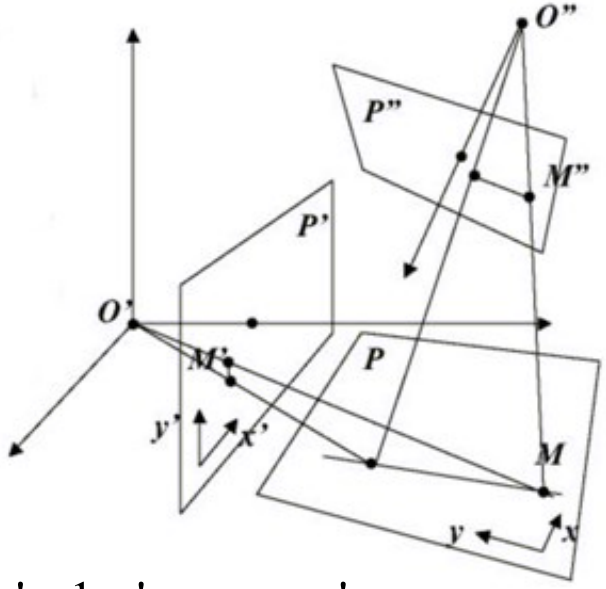
Geometric transformations



Homographies between two views

- Homography between a 3D plane and its image
- Homography between 2 images of the same 3D plane





$$x_i^{'} = H^{'} x_i$$
$$x_i^{"} = H^{"} x_i$$

$$\longrightarrow \quad x_i^{"} = H^{"} H^{' -1} x_i^{'} = H x_i^{'}$$

H, H' and H'' are homographies

x, x' and x'' points are in homogeneous coordinates

Homogeneous coordinates

(

Homogeneous coordinates

- Homogeneous coordinates are a key tool for
  - 3D computer vision
  - 3D computer graphics
  - Modeling robotic manipulators
  - …
- They allow us to transform between reference frames with a single matrix multiplication
  - in normal Euclidean coordinates,
    rotation is expressed by a matrix multiplication
    but translation is expressed by an addition …

Homogeneous coordinates

- (N+1)-dimensional notation for points in N-dimensional Euclidean space
- Allows us to express translation and projection as linear operations

Normal coordinates

$$v = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Homogeneous coordinates

$$v = \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

w is an arbitrary constant
$wx = w \cdot x$

Example:

$$v = \begin{bmatrix} 4 \\ -2 \\ 5 \end{bmatrix}$$

w = 1 → $v = \begin{bmatrix} 4 \\ -2 \\ 5 \\ 1 \end{bmatrix}$

w = -2 → $v = \begin{bmatrix} -8 \\ 4 \\ -10 \\ -2 \end{bmatrix}$

- To recover normal coordinates, divide de first N components by (N+1)th, w.

Homogeneous coordinates

- 3D rotation

Normal coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = [R] \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Some properties of matrix R
  - orthogonal matrix $\Rightarrow RR^T=I \quad \Rightarrow R^{-1}=R^T$
    - the dot product of any pair of rows or any pair of columns is zero
  - normalized matrix: the squares of the elements in any row or columns sum to 1
  - the rows of R represent the coordinates in the original space of unit vectors along the coordinate axis of the rotated space
  - the columns of R represent the coordinates in the rotated space of unit vectors along the coordinate axis of the original space

Homogeneous coordinates

- 3D (rotation + translation)

Homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_X \\ R_{21} & R_{22} & R_{23} & T_Y \\ R_{31} & R_{32} & R_{33} & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Using homogeneous coordinates,
  Rotation and Translation can be expressed by a single matrix

Homogeneous coordinates

- Perspective projection (along Z)

$$x' = \frac{f}{z}x \qquad y' = \frac{f}{z}y$$

Homogeneous coordinates

$$\begin{bmatrix} wx' \\ wy' \\ 0 \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- 3D scaling

Homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_X & 0 & 0 & 0 \\ 0 & S_Y & 0 & 0 \\ 0 & 0 & S_Z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Homogeneous coordinates

)

Homography

- Perspective projection of a plane
  - Other designations:
    - *texture-map, collineation, planar projective map*
  - Modeled as a 2D distortion (*warp*) using **homogeneous coordinates** (*see next slides*)

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- **Matrix H is defined up to an arbitrary scale factor and has therefore <u>only 8 independent entries</u>.**
  - so, 4 corresponding points are enough to compute the elements of matrix **H** as each pair of points gives rise to 2 independent equations

- Corresponding points in two views of the same world plane are also related by an homography (*see previous slide*).

Homography

- Determination of the matrix elements
  - establish the correspondences between <u>at least 4 points</u> belonging to 2 different planes
  - solve the following set of equations (for the chosen **<u>n</u>** points)

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2 x_2 & -x'_2 y_2 & -x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2 x_2 & -y'_2 y_2 & -y_2 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ x_N & y_N & 1 & 0 & 0 & 0 & -x'_N x_N & -x'_N y_N & -x'_N \\ 0 & 0 & 0 & x_N & y_N & 1 & -y'_N x_N & -y'_N y_N & -y'_N \end{bmatrix} \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \\ H_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ \dots \\ \dots \\ 0 \\ 0 \end{bmatrix}$$

A **H** **0**

2n x 9 **9** **2n**

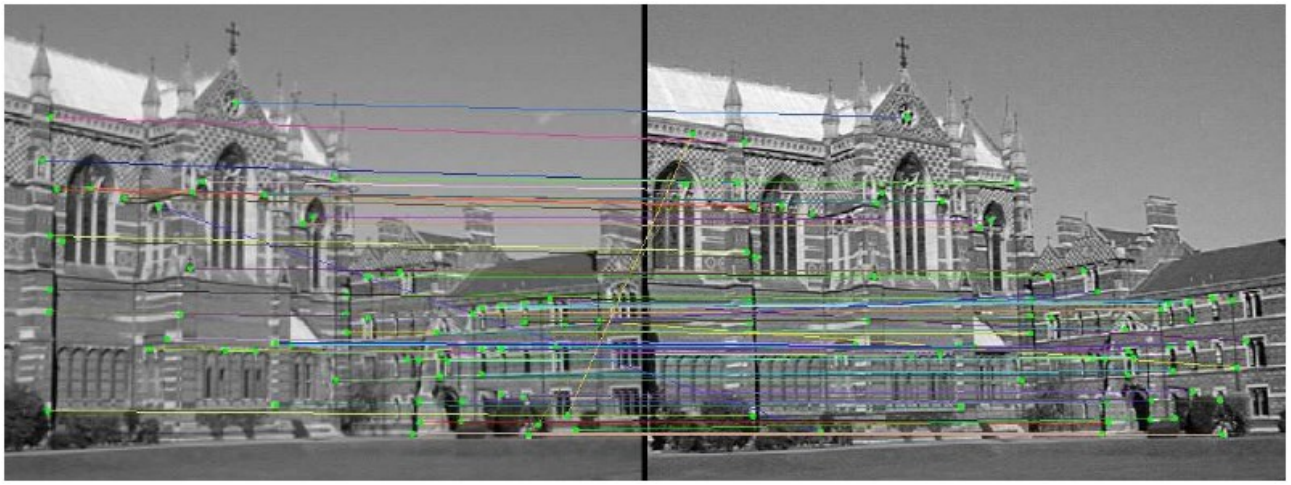
<u>Solution</u>: H = eigenvector of $A^T A$
corresponding to the smallest eigenvalue

Homography computation

- The computation of homographies must be robust enough to deal with
  - errors in the detection of some corners
  - wrong matches
  - occluded corners
- For that,
  the *RANSAC* (*RANdom SAmple Consensus*) may be used

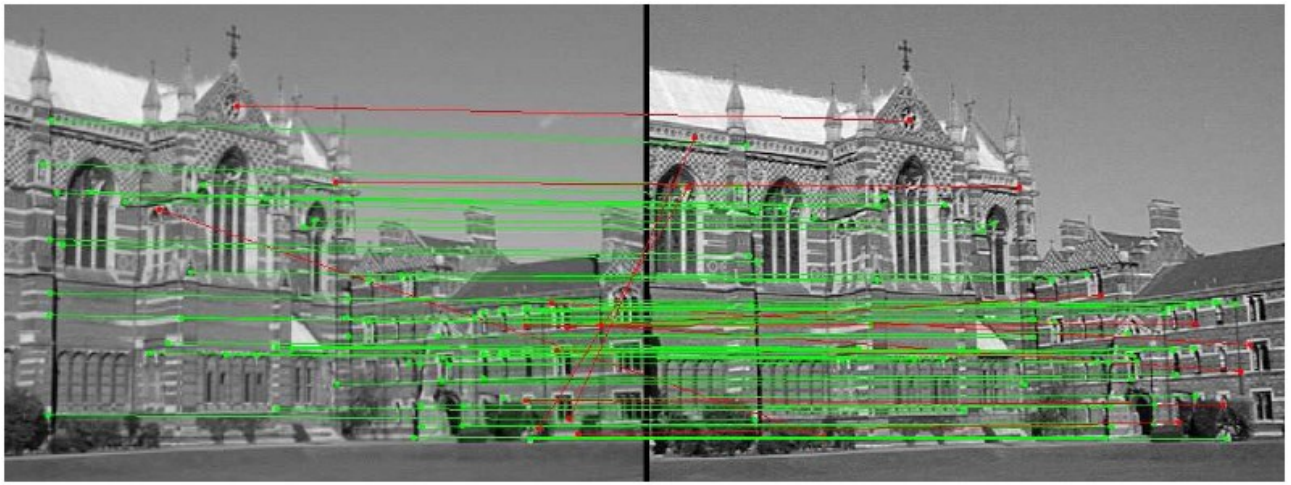
  (*M. A. Fischler, R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM, Vol 24, pp 381-395, 1981*)
  (*RANSAC for dummies, http://vision.ece.ucsb.edu/~zuliani/Research/RANSAC/docs/RANSAC4Dummies.pdf*)
  - this method allows a robust model fitting
    in the presence of many *outliers.*

RANSAC for refining matching



Initial matching results

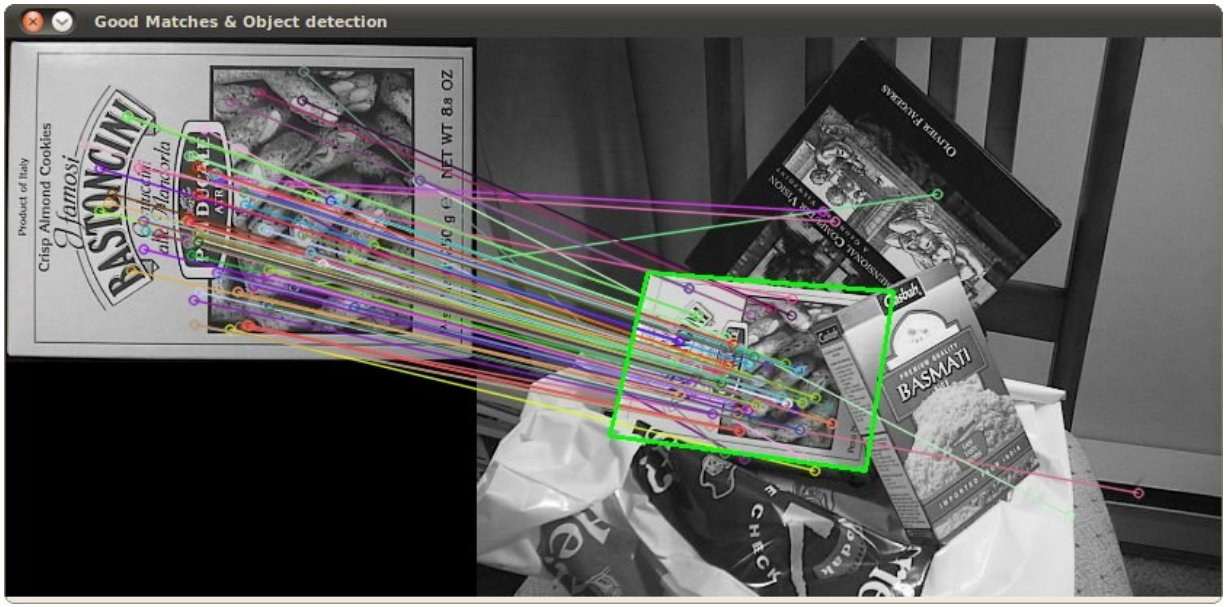
After RANSAC:
green points and lines represent inliers and red ones are outliers



Note: if the 2 images are acquired by simply rotating the camera (=> same center of projection) and the scene is very far away, the points in the 2 images are related by an homography !

Feature detection + Homography to find a known object

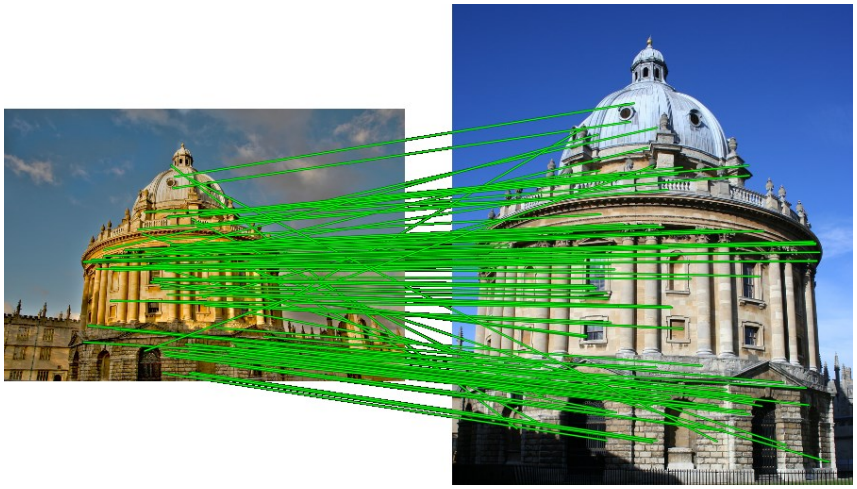
- Example from OpenCV site. Uses:
  - SURF extractor and descriptor
  - Flann-based matcher
  - Homography to map object corners to scene (and then draw the green rectangle)

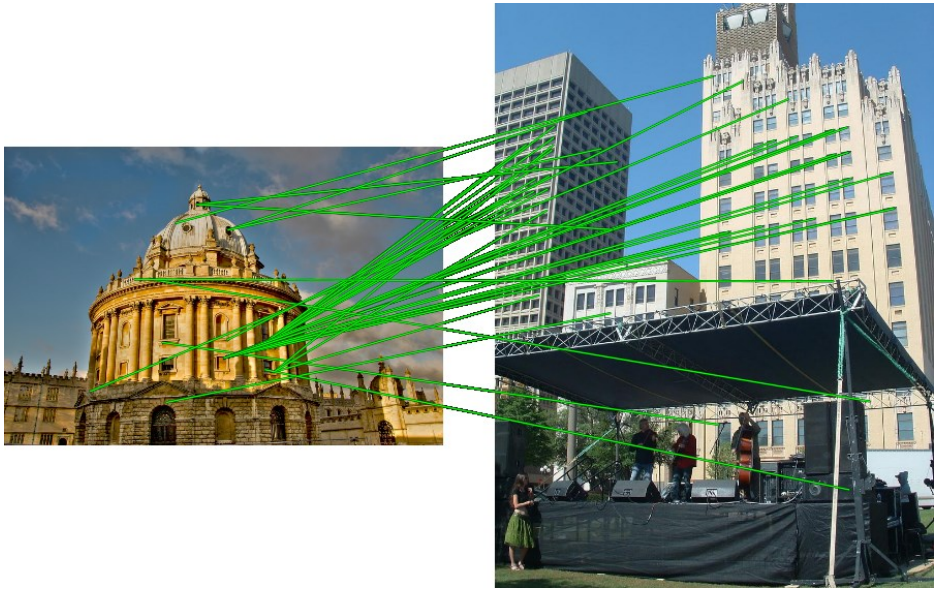


The detected object is highlighted in green

https://docs.opencv.org/3.4/d7/dff/tutorial_feature_homography.html

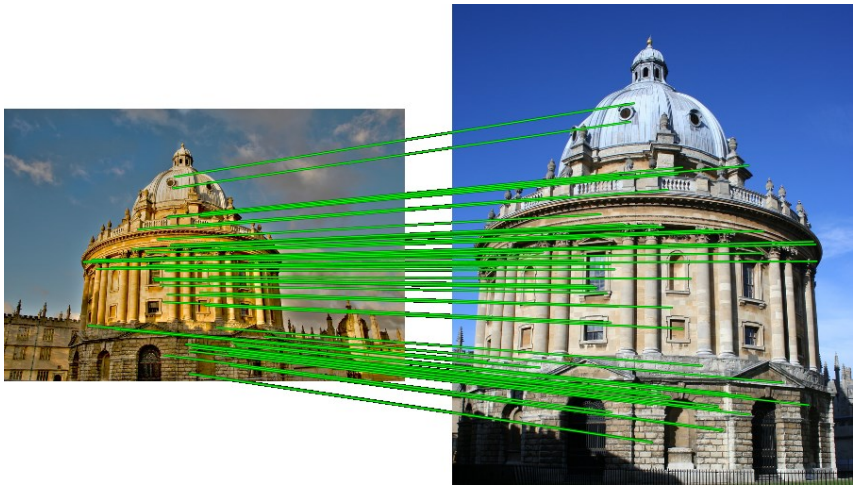
Image matching





Both image pairs have many matches.

Image matching



Only some of the matches are mutually consistent

We need to exclude all other matches (i.e. **outliers**)

RANdom SAmple Consensus
RANSAC

RANSAC

- RANdom SAmple Consensus [Fischler & Bolles,1981]

- **Approach**:
  - we want to avoid the impact of outliers,
    so let's look for "inliers", and use those only.

- **Intuition**:
  - if an outlier is chosen to compute the current fit,
    then the resulting transformation
    won't have much support from rest of the matches.

The RANSAC algorithm

- The problem:
  - given a fitting problem with parameters [$x_1..x_n$], estimate the parameters values
- Assume that:
  - the parameters may be estimated from *N* data values
  - there are a total of *M* data values
- The algorithm [Fischler & Bolles,1981]:
  - 1) Select, randomly, *N* samples
  - 2) Estimate the parameters [$x_1..x_n$] (obtain a model)
  - 3) Determine how many of the *M* samples fit the obtained model, given a tolerance; let *K* be the number of samples
  - 4) If *K* is large enough, accept the model parameters and end with SUCESS
  - 5) Repeat 1) to 4), *L* times
  - 6) If this step is reached, end with FAILURE
- L is difficult to be determined...
  Formule for determining *L :*

$$L = \frac{\log(p_{fail})}{\log(1 - p_g^N)}$$

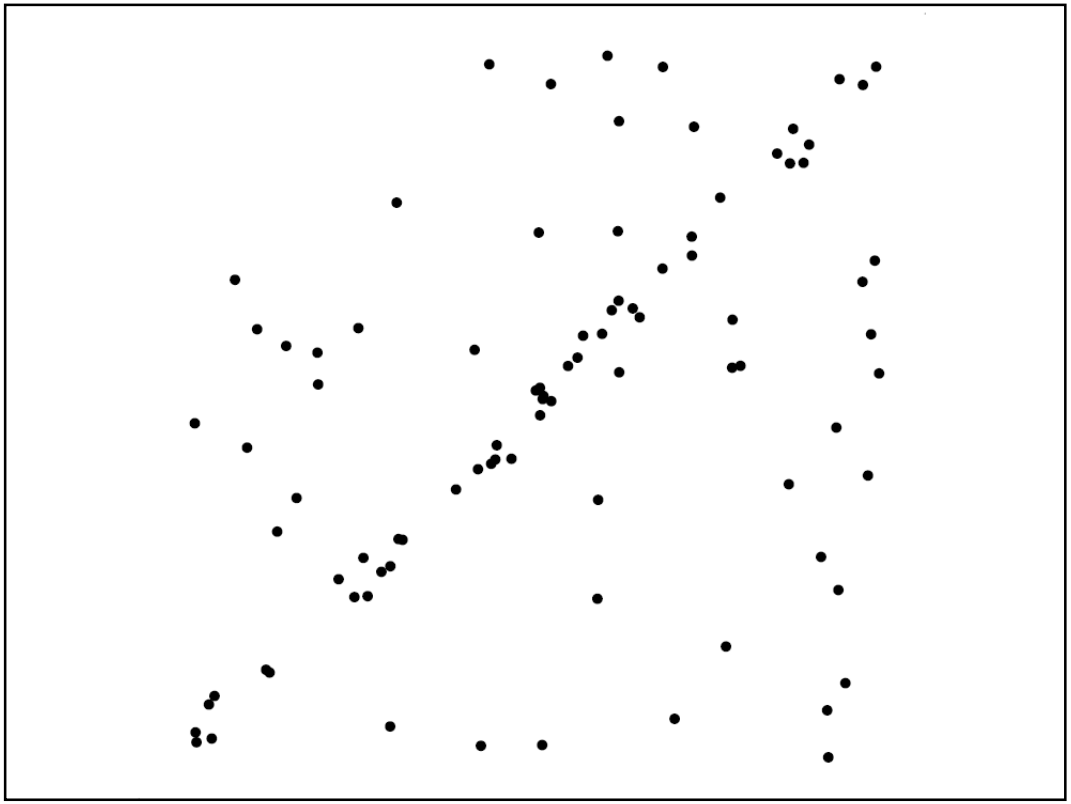
p_{fail} - probability of failing the selection of N *inliers* during *L* repetitions

p_g - probability of finding a good sample; in general, it is unknown at the beginning, being updated along the iterations

RANSAC for line fitting

- It is easy to understand the RANSAC concept using a line fitting example
- Repeat ***N*** times:
  - Draw 2 points at random
  - Fit line to these 2 points
  - Find inliers to this line among the remaining points (i.e., points whose distance from the line is less than ***t***)
  - If there are ***d*** or more inliers, accept the line and refit using all inliers

RANSAC for line fitting - example

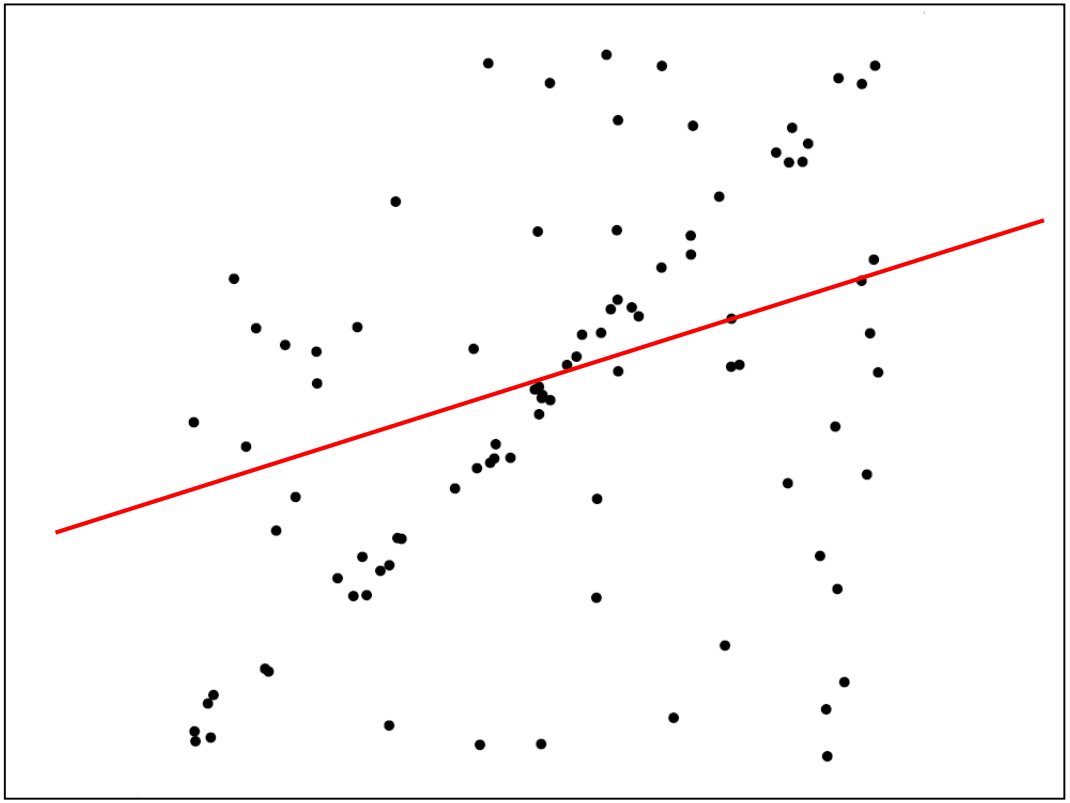


Source: R. Raguram

RANSAC for line fitting - example

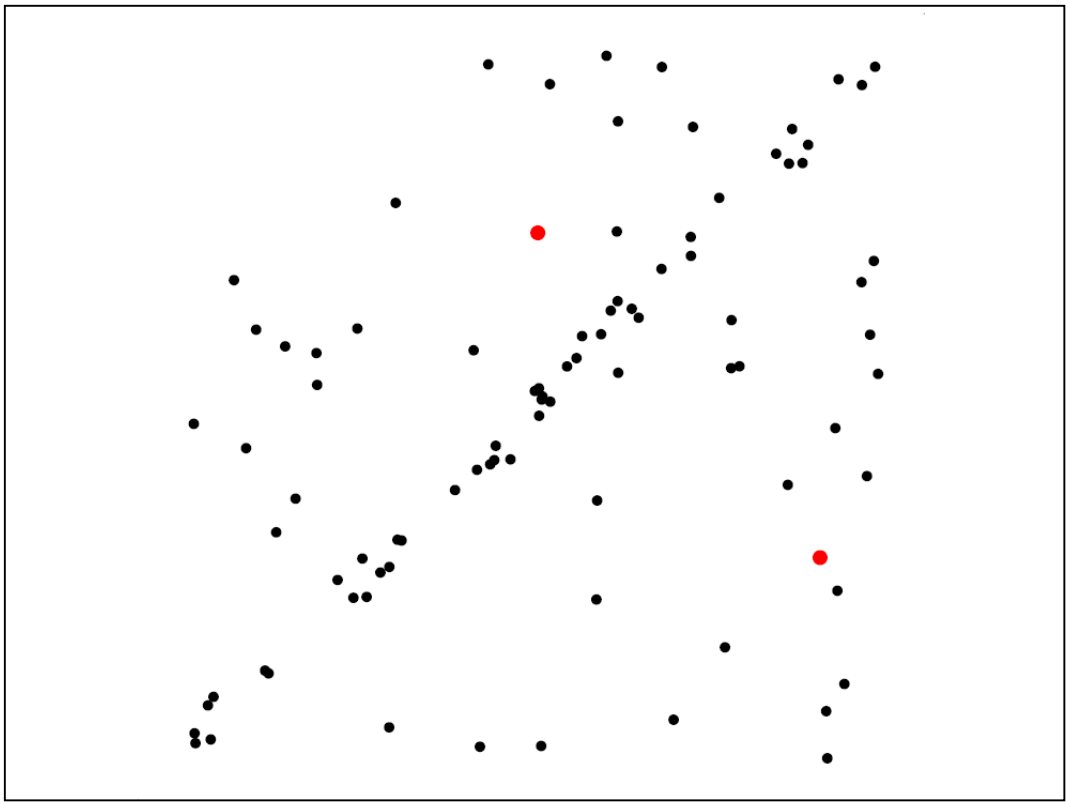
Least-squares fit:

- very poor result



Source: R. Raguram

RANSAC for line fitting - example

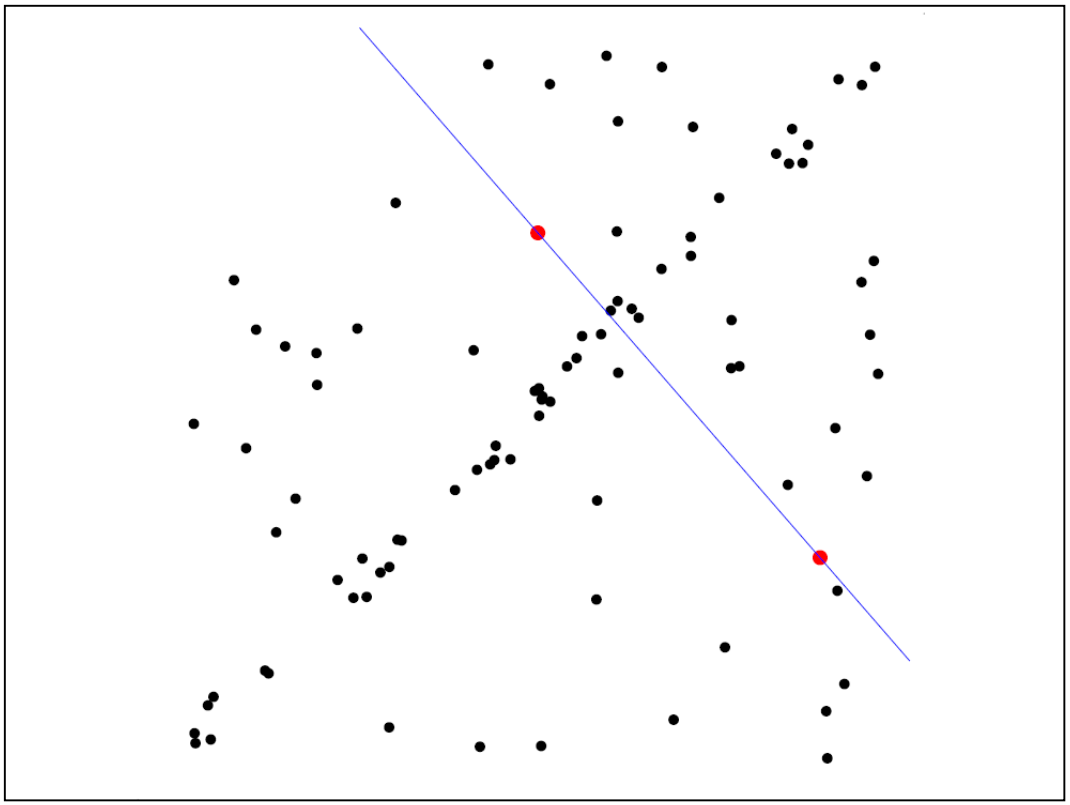


1. **Randomly select minimal subset of points**

Source: R. Raguram

Lana Lazebnik

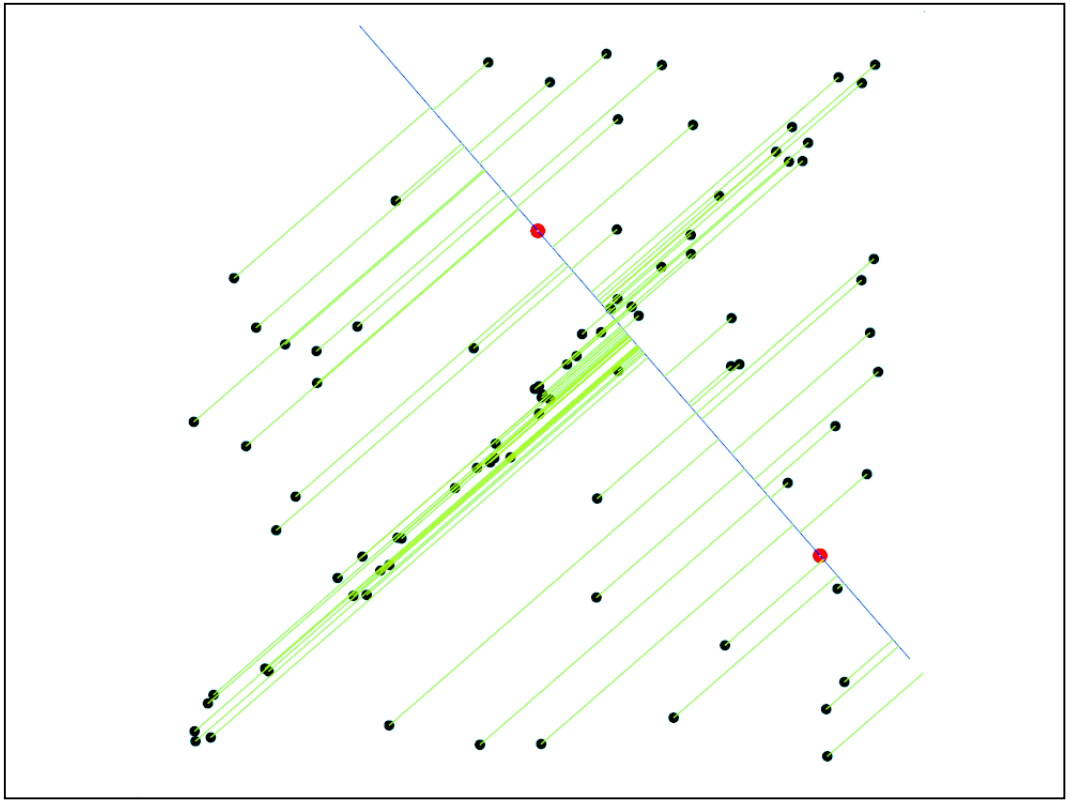
RANSAC for line fitting - example



1. **Randomly select minimal subset of points**
2. **Hypothesize a model**

Source: R. Raguram

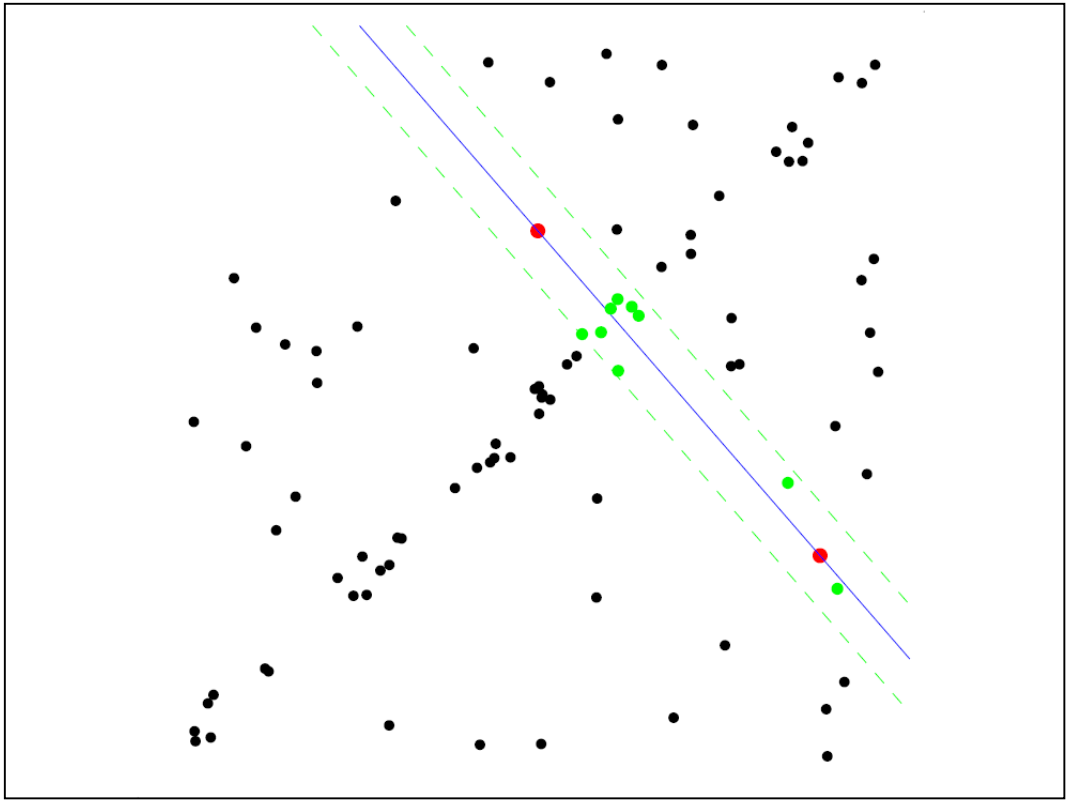
RANSAC for line fitting - example



1. **Randomly select minimal subset of points**
2. **Hypothesize a model**
3. **Compute error**

Source: R. Raguram

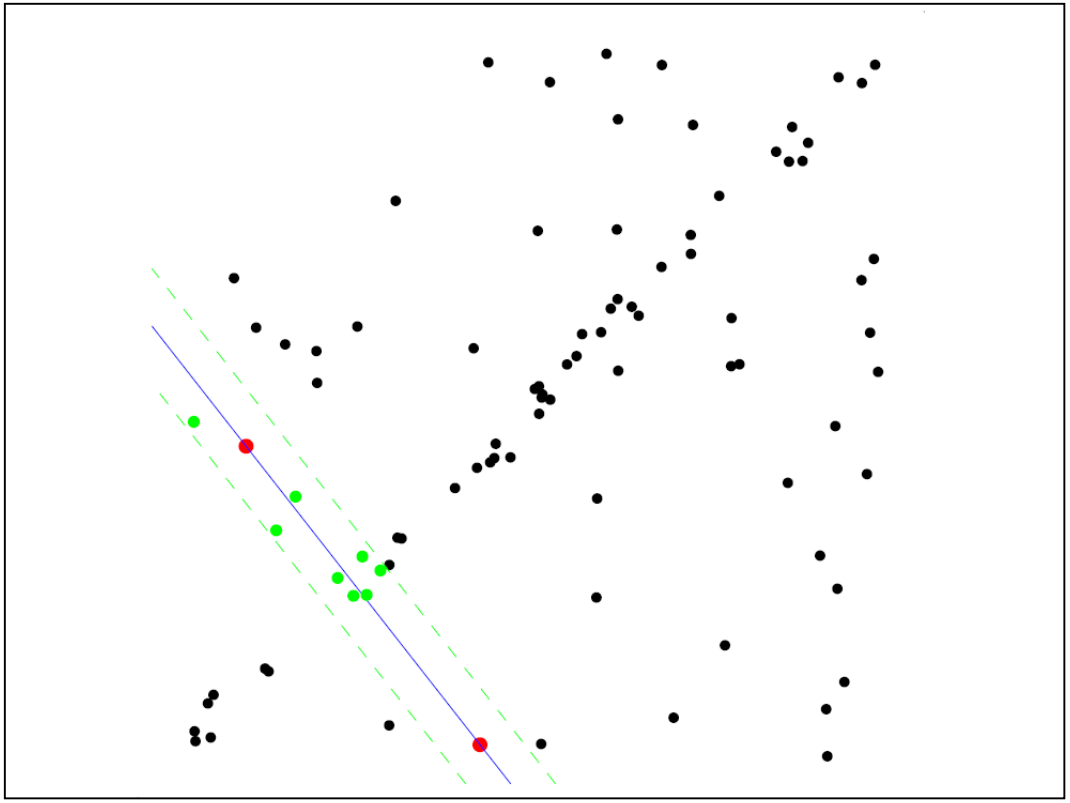
RANSAC for line fitting - example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error
4. Select points consistent with model

Source: R. Raguram

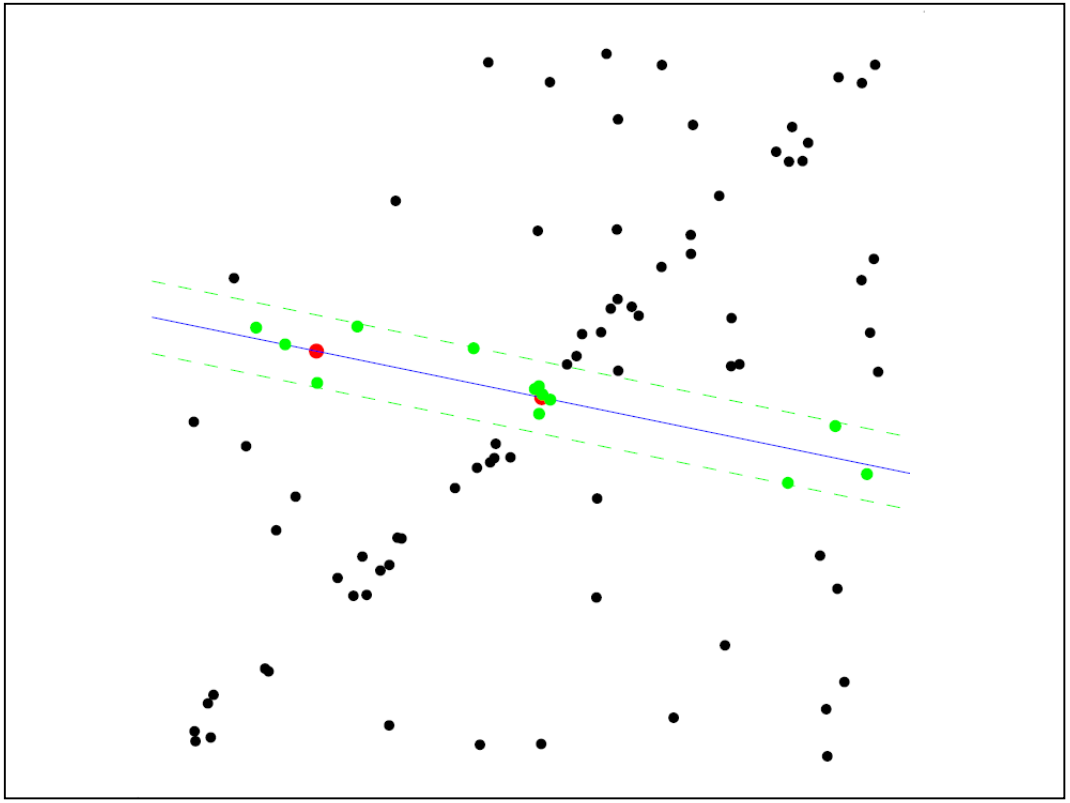
RANSAC for line fitting - example



1. **Randomly select minimal subset of points**
2. **Hypothesize a model**
3. **Compute error**
4. **Select points consistent with model**
5. **Repeat *hypothesize-and-verify* loop**

Source: R. Raguram

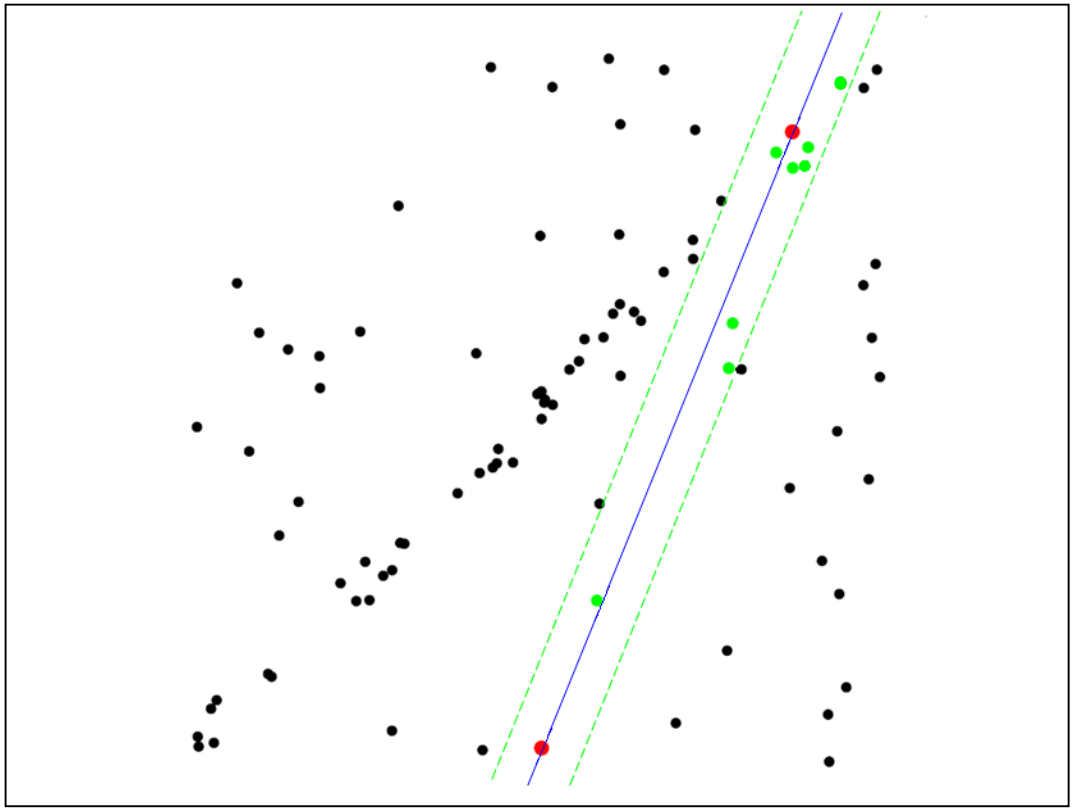
RANSAC for line fitting - example



1. **Randomly select minimal subset of points**
2. **Hypothesize a model**
3. **Compute error**
4. **Select points consistent with model**
5. **Repeat *hypothesize-and-verify* loop**

Source: R. Raguram

RANSAC for line fitting - example



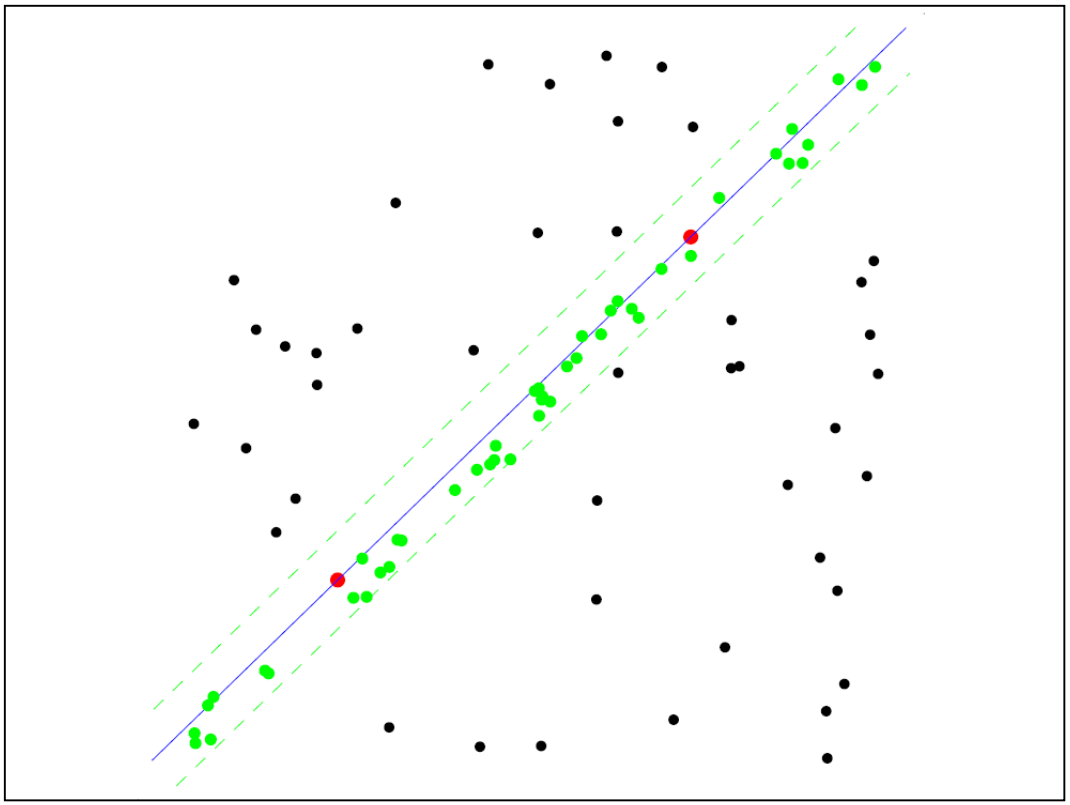
1. **Randomly select minimal subset of points**
2. **Hypothesize a model**
3. **Compute error**
4. **Select points consistent with model**
5. **Repeat *hypothesize-and-verify* loop**

Source: R. Raguram

Lana Lazebnik

RANSAC for line fitting - example

Uncontaminated sample



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

Lana Lazebnik

The RANSAC algorithm

- That was an example fitting a line
- What about fitting a transformation (translation, homography, …)?

RANSAC: general form

RANSAC loop:

1. Randomly select a *seed group* on which to base transformation estimate (e.g., a group of matches)
2. Compute transformation from seed group
3. Find *inliers* to this transformation
4. If the number of inliers is sufficiently large, re-compute estimate of transformation on all of the inliers

- Keep the transformation with the largest number of inliers

Homography calculation using *RANSAC*

- Algorithm:
  1. select, randomly, 4 non-collinear, matched points in two images of the same marker or in the marker and in an acquired image of the marker: $\{ [x\ y\ 1]^T \leftrightarrow [x'\ y'\ 1]^T \}$
  2. calculate the homography, H, using this random sample
  3. for all the matched points, calculate the distance between $[x'\ y'\ 1]^T$ and $[H]\ [x\ y\ 1]^T$
  4. count the number of matched pairs for which the distance is below a threshold; these pairs are considered as *inliers* (I_H=no. of *inliers*) and the remaining pairs are considered as *outliers* (*)
  5. a maximum number of iterations was reached STOP and EXIT with FAILURE
  6. if the number of *outliers* is very high or the ratio *outliers/inliers* is above a threshold, go to step 1,
  7. recalculate the homography using the *inliers*

(*) Note: the reprojection error (distance between a projected point and a measured one) is usually used as an additional criterion to treat a point pair as an inlier

RANSAC "song"

- When you have outliers you may face much frustration
  if you include them in a model fitting operation.
  But if your model's fit to a sample set of minimal size,
  the probability of the set being outlier-free will rise.
  Brute force tests of all sets will cause computational constipation.
- *N* random samples
  will provide an example
  of a fitted model uninfluenced by outliers. No need to test all combinations!
- Each random trial should have its own unique sample set
  and make sure that the sets you choose are not degenerate.
  N, the number of sets, to choose is based on the probability
  of a point being an outlier, and of finding a set that's outlier free.
  Updating *N* as you go will minimise the time spent.
- So if you gamble
  that *N* samples are ample
  to fit a model to your set of points, it's likely that you will win the bet.
- Select the set that boasts
  that its number of inliers is the most (you're almost there).
  Fit a new model just to those inliers and discard the rest,
  an estimated model for your data is now possessed!
  This marks the end point of your model fitting quest

RANSAC pros and cons

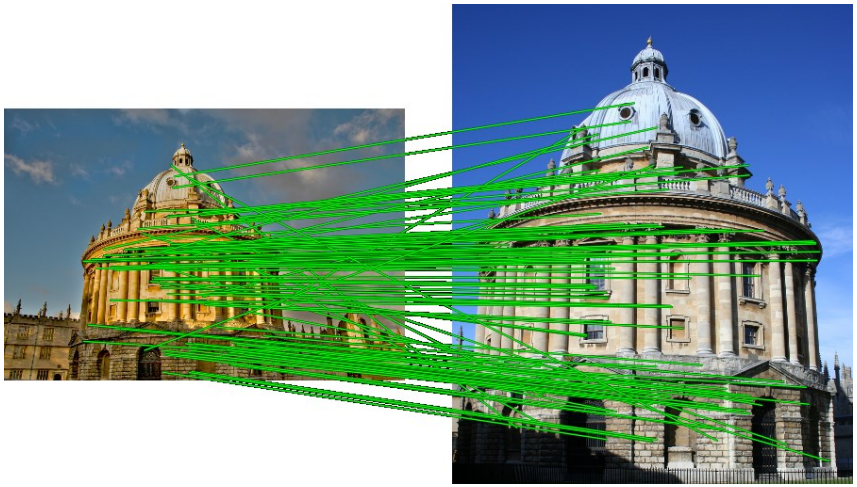
- Pros
  - Simple and general
  - Applicable to many different problems
  - Often works well in practice
- Cons
  - Parameters to tune
  - Doesn't work well for low inlier ratios (too many iterations, or can fail completely)
  - Can't always get a good initialization of the model based on the minimum number of samples

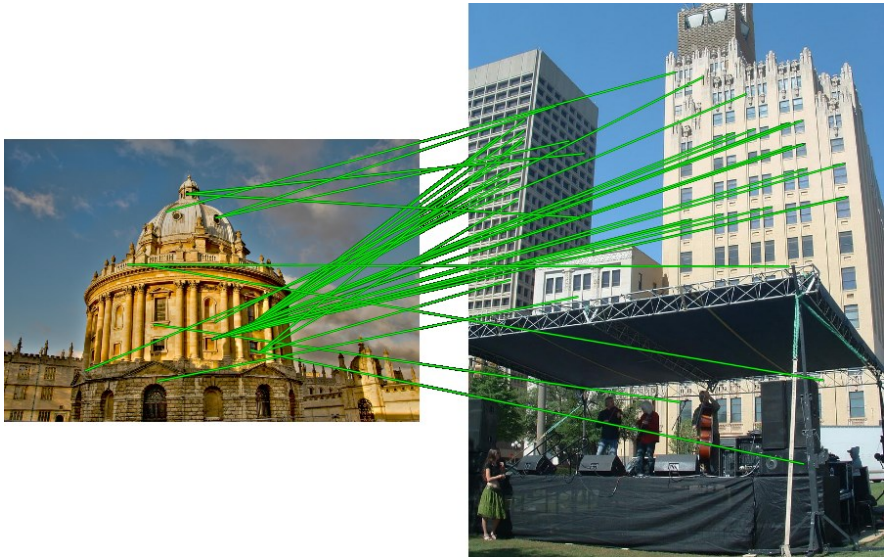
slide credit: Lana Lazebnik

RANSAC

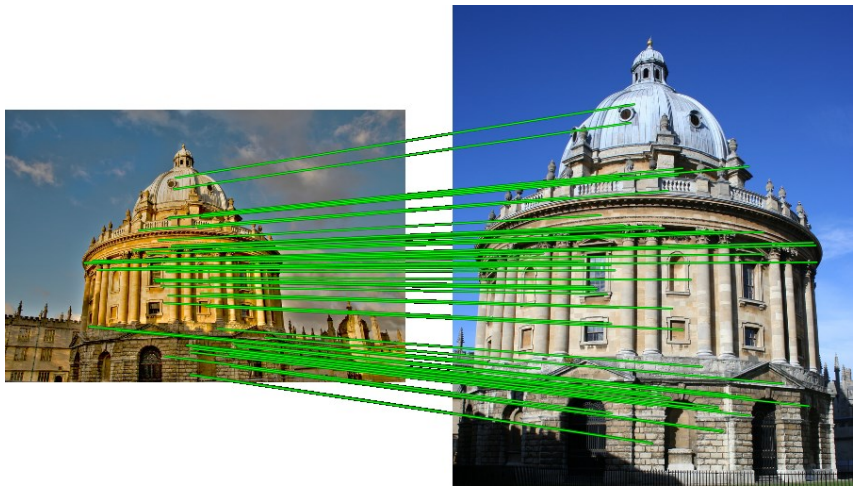
)

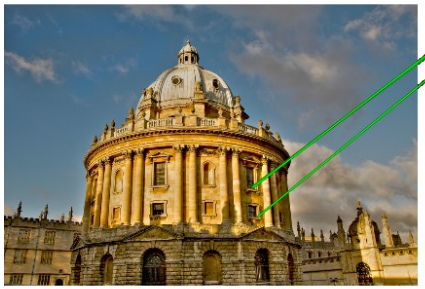
No verification





RANSAC verification

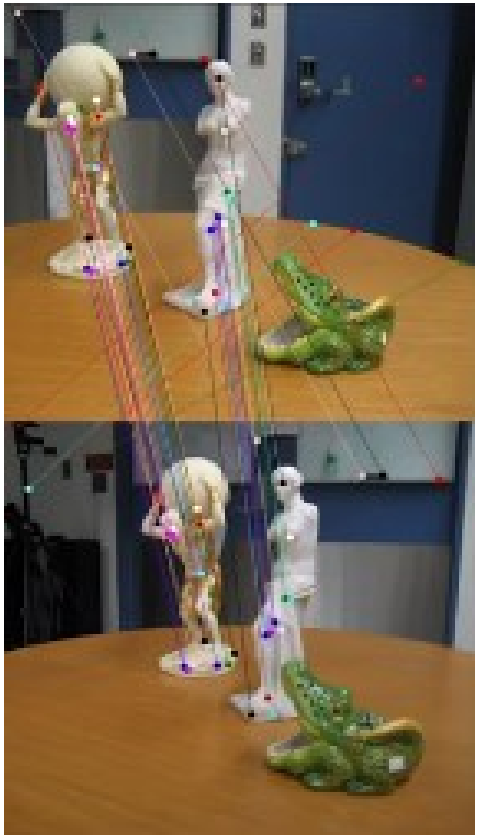




Fails to meet
threshold on #inliers

SIFT for 3D objects?

No change in viewpoint



30º change in viewpoint

90º change in viewpoint

slide: adapted from Shree Nayar, Columbia University

OpenCV
feature detection, description & matching

Feature detectors:
- SIFT
- SURF
- MSER
- FAST
- BRISK
- ORB
- AKAZE
- KAZE
- STAR
- MSD

Feature descriptors:
- SIFT
- SURF
- BRIEF
- BRISK
- ORB
- KAZE
- AKAZE
- FREAK
- DAISY
- LATCH
- LUCID

Feature matchers:
- BF
- FLANN

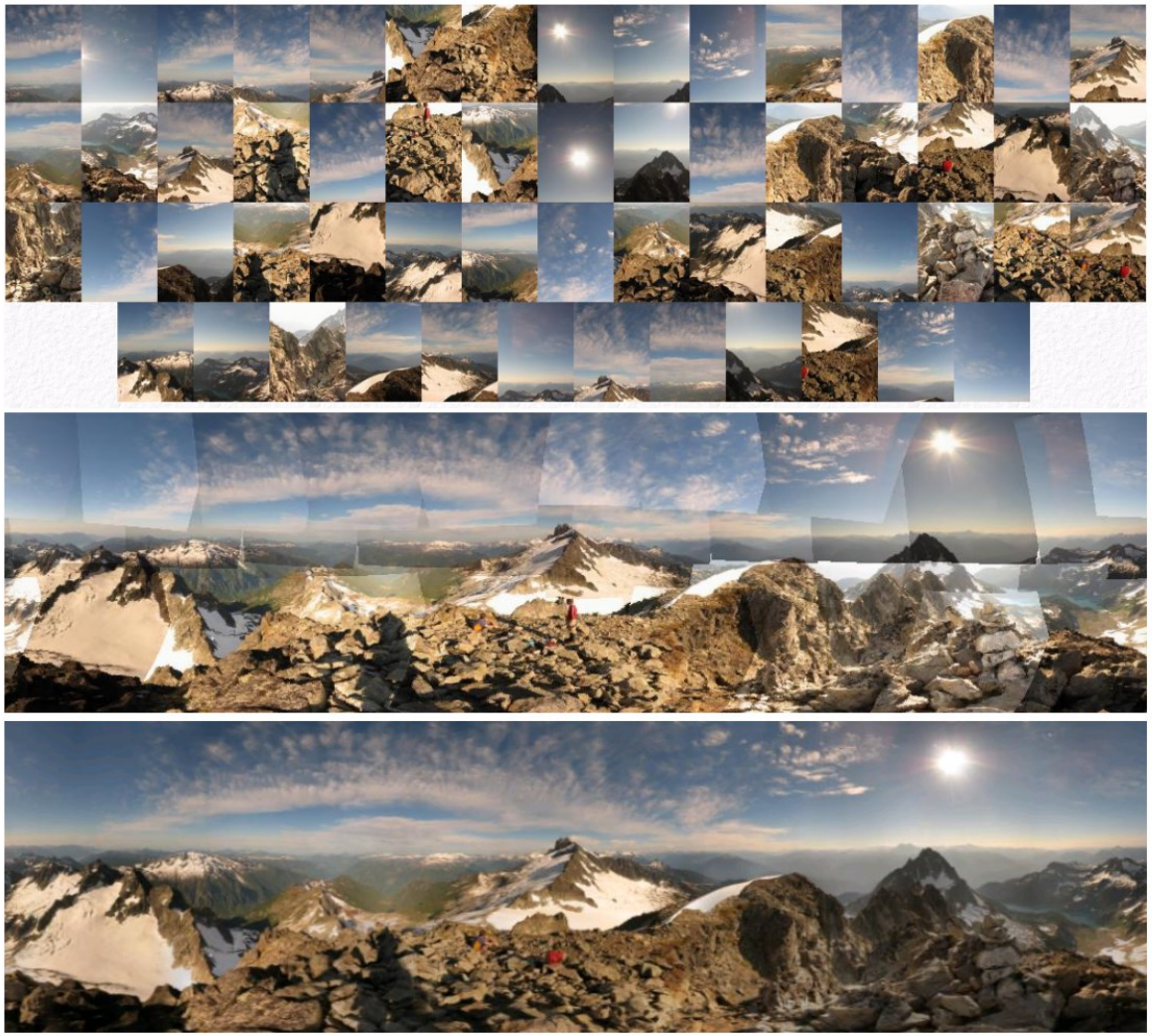
Correspondence rejection:
- Lowe's ratio
- RANSAC

Feature detection & matching: applications

- Recognition
- Panoramas
- Motion tracking
- Mobile robot navigation
- 3D reconstruction
- Wide baseline stereo
- …

Applications

Automatic mosaicing



http://matthewalunbrown.com/autostitch/autostitch.html